ARI Research Note 96-55

Problem Solving and Learning in a Natural Task Domain

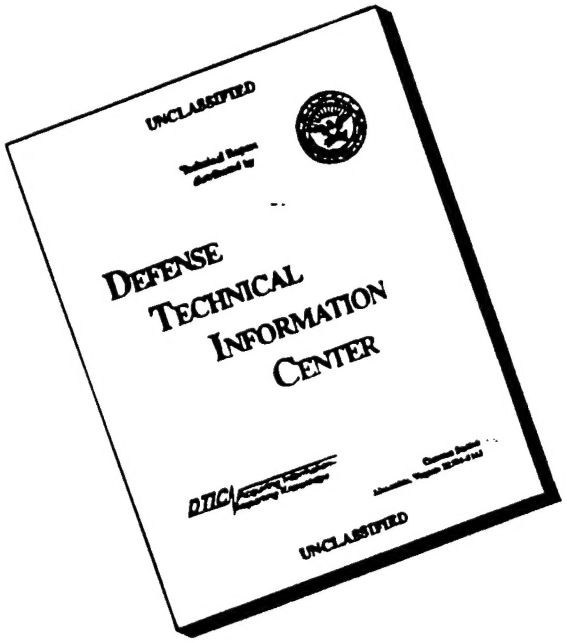
Janet Kolodner and Lawrence Barsalou
Georgia Institute of Technology

Research and Advanced Concepts Office
Michael Drillings, Acting Director

March 1996

United States Army
Research Institute for the Behavioral and Social Sciences

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

U.S. ARMY RESEARCH INSTITUTE FOR THE BEHAVIORAL AND SOCIAL SCIENCES

A Field Operating Agency Under the Jurisdiction of the Deputy Chief of Staff for Personnel

EDGAR M. JOHNSON
Director

Research accomplished under contract
for the Department of the Army

Georgia Institute of Technology

Technical review by

Joseph Psotka

REPORT DOCUMENTATION PAGE

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED (from. . . to)
1996, March	Final	September 1986-August 1989

4. TITLE AND SUBTITLE	5a. CONTRACT OR GRANT NUMBER
Problem Solving and Learning in a Natural Task Domain	MDA903-86-C-0173
	5b. PROGRAM ELEMENT NUMBER 0601102A
6. AUTHOR(S) Janet Kolodner and Lawrence Barsalou (College of Computing--Georgia Institute of Technology)	5c. PROJECT NUMBER B74F
	5d. TASK NUMBER BR41
	5e. WORK UNIT NUMBER C71
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Georgia Institute of Technology College of Computing Atlanta, GA 30332-0280	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) U.S. Army Research Institute for the Behavioral and Social Sciences ATTN: PERI-BR 5001 Eisenhower Avenue Alexandria, VA 22333-5600	10. MONITOR ACRONYM ARI
	11. MONITOR REPORT NUMBER Research Note 96-55

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

COR: George Lawton

14. ABSTRACT *(Maximum 200 words)*:

This report explains the details of some of the problem solving and learning processes employed by novice problem solvers as they become more expert. In particular, the researchers investigate the effects of individual problem solving and learning experiences on later problem solving in the context of troubleshooting.

15. SUBJECT TERMS

Problem solving Symptom-fault pairs Learning Troubleshooting

SECURITY CLASSIFICATION OF			19. LIMITATION OF ABSTRACT	20. NUMBER OF PAGES	21. RESPONSIBLE PERSON (Name and Telephone Number)
16. REPORT	17. ABSTRACT	18. THIS PAGE			
Unclassified	Unclassified	Unclassified	Unlimited	185	

Final Report
Problem Solving and Learning in a Natural Task Domain
MDA-903-86-C-173

Janet Kolodner
Lawrence Barsalou
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
jlk@cc.gatech.edu

Our goal in this project has been to explain the details of some of the problem solving and learning processes employed by novice problem solvers as they become more expert. In particular, we have investigated the effects of individual problem solving and learning experiences on later problem solving in the context of troubleshooting.

In year 1 of the project, we collected and analyzed protocols of students solving several sequences of diagnostic problems in the domain of car mechanics. A theoretical analysis of these protocols led us to several working hypotheses about problem solving and learning (Lancaster & Kolodner, 1987, 1988).

First, we found three types of knowledge necessary for diagnosis.

- The **qualitative causal model** provides knowledge about what system behaviors derive from other system behaviors or states. It is eventually made up of two parts: the model of how the system *works* and the model of how it *malfunctions.*
- **Symptom-fault pairs** provide associational knowledge that associates symptoms and other contextual factors with potential faults and allows the reasoner to *index* into the causal model.
- **Reasoning strategies** provide knowledge about what actions to take in solving a problem.

Together, the three types of knowledge make up a reasoner's *mental model* of a device. The first two types of knowledge tell the reasoner how the device works. The third type of knowledge tells him how to use the first two types in order to solve problems or troubleshoot.

The knowledge and organization of these different kinds of knowledge changes with experience. In the causal model, there are two notable changes that happen through experience. First, connections between parts of the model are learned. The novice, for example, is clearly unaware of the possibility that electronic failures can affect things like fuel delivery, since he knows little about the dependencies between the fuel system and the electrical system, while the more advanced mechanic not only knows that such relationships exist, he considers them a highly common source of failures. Second, the organization of the model becomes more hierarchical as students learn which parts are subsumed by or part of other systems. While initial representations of the causal model seem to be flat (our interpretation of the way novices went about solving problems), after considerable experience, the model becomes a more complex, hierarchical system. A novice car mechanic, for example, might consider the carborator of a car on equal footing with every other part of the engine, and in diagnosing a particular problem might consider each part of the engine individually. More advanced mechanics, on the other hand, seen to organize the parts of the engine into systems. The data suggest that for experienced mechanics, components are organized hierarchically under their respective systems and are never directly considered unless their system is determined to house the failure, or at least to be the source of information crucial to locating the failure. Similarly, the number, organization, and accuracy of the symptom-fault sets changes with increasing experience.

Building partly on these changes in the knowledge structures, and partly on independent effects of experience on decision processes, the mechanic's reasoning strategies also change. For example, procedures and guidelines for accepting hypotheses as diagnoses become increasingly focussed on information that allow a causal interpretation of the behavior observed. At the same time, the developing knowledge structures allow the mechanic to search for and aquire more, and more accurate, information from his symptom-fault sets and his causal model.

The interaction of these changes in both knowledge and process lead to the more accurate and efficient problem solving seen in experts. His overall level of knowledge is increasing; the organization and integration of his knowledge structures, both the symptom-fault sets and the causal model, are increasing; and his processes and criteria for reaching diagnoses are becoming more accurate, more efficient, and more focussed on causal information.

Based on analysis of the same protocols, we have identified five different real-world learning procedures used by the students and several roles a teacher or more experienced third party must play in helping a student to learn at various stages.

- **Learning by understanding explanations** is the process by which students integrate a teacher's explanation of how to solve a problem with their own diagnostic knowledge.

- **Active gap filling** allows a student to fill in known gaps in his knowledge by asking questions or looking in source books.

- **Learning from interpreting feedback** is used when the student is unable to evaluate test results he has obtained. Interpretation may or may not require intervention of a teacher, depending upon the student's knowledge state.

- **Abstraction** lets the student reorganize his knowledge in better ways. It can be done independently by a student or pointed out by a teacher.

- **Case-based reasoning** allows the student to improve his problem solving without having a full understanding of how things work, and guides students in the building of abstractions. The cases students remember might be their own attempts to solve problems or the explanations given by a teacher of how to solve a problem. We have identified three circumstances under which students seem to retain their experiences: when an experience was unexplainable, when it provided the first introduction to some concept,and when a serious mistake was made.

Appendix A explains this work in more detail.

Later work has followed up on these observations and hypotheses, both through experimentation and through the implementation and evaluation of cognitive models of problem solving and learning activities.

Work on the psychology side in Years 2 and 3 was on three tasks:

- continuing the theoretical analysis of the first year in an attempt to describe it more rigorously, and in particular, to come up with a set of dimensions useful for assessing observed learning,

- building an instruction tool called MECH that can be used to run experiments to find out more detail about what people are learning and what instructional methods work best in teaching those things,

- deriving experimental techniques and experiments that will allow us to rigorously describe both the learning students do and the contexts in which they do it.

Our implementations of cognitive models on the computer has focused on providing more detail about the learning processes and representational structures suggested by the protocols. Work has been in four areas:

- an in-depth investigation of *learning by understanding explanations*, a learning process in which the student integrates what the teacher presents into his/her current mental model, implemented in two programs – EDSEL1 and EDSEL2,

- investigation of the *case-based reasoning* processes employed during problem solving, implemented in a program called CELIA,
- the creation of a representation of the knowledge used in troubleshooting that integrates general and case knowledge and that supports all of the diagnosis and learning procedures we have been studying,
- the creation of memory models that integrate the three different kinds of knowledge problem solvers use and that support both learning processes that we have been investigating, implemented in several different versions of a program called CORA.

We explain each in turn, referring to published papers in the appendices of this report for those parts of the research that have been published.

1 Mental Models: Representation and Use

The protocol studies we did in year one provided several hypotheses for us about the structure of mental models, how their pieces can be accessed, and what needs to be learned to use them well. We followed up that study with a study of the literature, and based on the combination of protocols and literature, we have been able to derive a more substantial and rigorous description of some of the knowledge used in reasoning about devices. We have also come up with dimensions for assessing whether and how well this knowledge has been learned.

1.1 Principles of Mental Models

We have found that the following representational assumptions and entities go a long way in accounting for the structural aspects of mental models. First, any mental model is based on the following three representational assumptions:

1. ANALOGICAL MAPPING: There is generally a 1:1 mapping of components and relations in a physical device to representations of components and relations in a mental model (Johnson-Laird, 1983).
    - CAVEAT 1A: Not all components and relations in a physical device may be represented in a mental model.
    - CAVEAT 1B: Components and relations may not be correctly represented.

2. HIERARCHICAL ORGANIZATION: To the extent the organization of a physical device is hierarchical, the organization of a mental model will be hierarchical. More specifically, the device and the mental model both decompose to subsystems, which in turn decompose to more specific subsystems, etc., before decomposing to terminal components. Components belonging to two subsystems may occasionaly violate the strict hierarchical organization of components. Even with these violations, it is still possible to decompose a device in a quasi-hierarchical manner that serves as a useful organization of components. In a small engine, decomposition may procede from the engine to the ignition and fuel systems, from the ignition system to the magneto and spark plug, from the magneto to the breaker points and coil, from the breaker points to terminal components such as the stationary point and moving point.

3. MULTIPLE MODELS: A given mental model may be one of many possible for the same physical device. There are many ways models can differ. There can be be simple ways in which different people represent the same device in slightly different ways, with there being more similarity than dissimilarity between models. On the other, there may be multiple models for the same device that capture fundamentally different kinds of I/O relations and serve fundamentally different kinds of reasoning goals (e.g., White and Fredericksen, 1986).

Any hierarchically-organized mental model contains the following representational entities:

- COMPONENTS: These include representations of the specific subsystems that compose a more general subsystem, as well as the terminal components of the most specific subsystems. In a small engine, components of the engine include the fuel system and ignition system (decomposition of a general subsystem to more specific subsystems); components of the coil include the primary wire and secondary wire (decomposition of a subsystem to terminal components).

- INTERNAL RELATIONS: These are representations of the input/output relations between components of a subsystem. In the ignition system, these include passage of current from the magneto to the breaker points to the spark plug; in the fuel system, these include passage of fuel from the tank to the fuel pickup pipe to the carburetor.

- EXTERNAL RELATIONS: These are representations of input/output relations from one subsystem to another. External relations are only possible when a hierarchical decomposition of a mental model exists. Note that external relations in some sense provide violations of hierarchical structure, since they criss-cross the decomposition hierarchy in ways that violate class inclusion. In a small engine, these include passage of current from the ignition system to the cylinder assembly and displacement of the drive train by the cylinder assembly.

Another kind of entity related to mental models seems quite interesting and important. It is similar to what Murphy and Medin (1985) and Schank, Collins, and Hunter (1986) have argued structures categories:

- GENERAL PHYSICAL MECHANISMS: These are representations of general physical mechanisms that in some way help integrate (at least initially) the components and relations of mental models. Examples of such mechanisms include the generation and amplification of electrical energy, the combustion of air-fuel mixture, the build-up and dissipation of heat, lubrication, and so forth. One way to think about these mechanisms is as very abstract input/output relations between very abstract components. Once these mechanisms are understood, they become mapped into more specific applications to help produce mental models. An interesting question is whether acquiring them can precede acquiring a mental model for a physical system or whether they are more easily acquired after having gained knowledge of how at least one physical system works. It is possible that the acquisition of a new mechanism may cause a person to fundamentally restructure a mental model (Collins, Salter, & Tenney, 198?).

1.2 Processing Mental Models

We have found that three kinds of knowledge account for much of the reasoning we see people doing about mental models. They are:

1. QUALITATIVE REASONING KNOWLEDGE: This knowledge specifies how the outputs from one component determine the outputs of another. People use this knowledge to simulate performance of the device. Series of these rules may be applied to see how an input to one component produces effects over paths of relations that emanate from the component. Following Hegerty et al. (1988), the operation of these rules depends on the properties of the the respective components, as well as on the inputs they receive (e.g., the rate of fuel flow through a tube depends on its diameter, as well as on the amount of fuel it receives as input). These properties and inputs provide constraints on the behavior of components. Qualitative reasoning knowledge captures these constraints and allows qualitative prediction about performance. More specific forms of this knowledge may allow quantitative prediction.

2. SYMPTOM-FAULT ASSOCIATIONS: These rules start with observed problematic symptoms and provide hypotheses about what components might be at fault. For a small engine, a symptom-fault rule might state that whenever there's a strong gas smell during engine operation, there's a good chance the choke is broken. A more specific symptom-fault rule might state that if, in addition to a strong gas smell, the engine type is Briggs and Stratton, the condenser might be broken.

3. REASONING STRATEGIES: These include guidelines about how the topology of a mental model should be searched to find a fault (e.g., breadth-first versus depth-first); rules about the transitivity of qualitative reasoning rules (e.g., if component X produces an input to component Y, and if component Y produces an input to component Z, then X produces the input to Z); rules about how to handle remindings; etc. In contrast to qualitative reasoning and symptom-fault knowledge, reasoning strategies are fairly domain independent. In general, reasoning strategies guide the executive control of troubleshooting by setting goals, deciding how to handle errors, handling interuptions and unexpected results, etc. (cf. Norman & Shallice, 1986).

An organizational principle also seems important to processing:

- COMPILATION: With practice at using any sequence of inferences repeatedly, the sequence may become compiled into a procedure that produces more efficient processing in the future. Sequences of qualitative reasoning rules may become automated for frequent kinds of qualitative reasoning. Sequences (or simultaneous sets) of symptom-fault rules may become automated to zero in quickly on suspected faults. Sequences of meta-rules may become automated to minimize wasted resources and non-optimal behavior. Moreover, combinations of different types of rules may become automated to the extent they frequently occur in a systematic pattern.

1.3 Learning Mental Models

There appear to be two important kinds of learning that can occur for mental models: (1) learning a mental model and (2) learning to use a mental model for troubleshooting a physical device. Learning mental models is addressed in this section; learning how to troubleshoot is addressed in the next.

Learning a mental model can be assessed on the following dimensions:

1. ACQUISTION OF COMPONENTS: To what extent are the components of the physical device represented in the mental model? What do these representations look like (content and structure)? How do they change over time? By what processes do they change? What triggers these changes?

2. ACQUISITION OF HIERARCHICAL STRUCTURE: To what extent is the hierarchical organization of components in a physical device represented in the hierarchical organization of components in the mental model? How does that change over time? By what processes? What triggers these changes?

3. ACQUISITION OF INTERNAL RELATIONS: Assuming components are hierarchically organized in a mental model, to what extent are the components within a particular subsystem integrated by the appropriate input/output relations? What does the representation look like (content and structure)? How does it change over time? By what processes? What triggers these changes?

4. ACQUISITION OF EXTERNAL RELATIONS: Assuming subsystems are hierarchically organized in a mental model, to what extent are they integrated by the appropriate input/output relations? What is the content of the integration (i.e., what relations are there)? How does it change over time? By what processes? What triggers change?

5. ACQUISITION OF GENERAL PHYSICAL MECHANISMS: To the extent that general physical mechanisms are important to properly integrating the components of a mental model, to what extent are these mechanisms represented and integrated with the model? What does the representation look like? How does it change over time? By what processes? What triggers change?

6. ACQUISITION OF MULTIPLE MENTAL MODELS: To the extent that fundamentally different models can usefully represent the same device, to what extent have they been represented and integrated? What does the integration look like? By what processes does the integration happen? How does it change over time? What triggers change?

The mental model includes both a model of how the device functions normally and a model of how it malfunctions. Questions above must be applied to both of those models. In addition, we must find out how these two models are integrated with each other? By what processes? How that integration changes over time? What triggers these changes?

1.4 Learning to Troubleshoot

Learning how to use a mental model during troubleshooting can be assessed on the following five dimensions:

1. USE OF REMINDINGS: To what extent do people use previous problem solving episodes to solve new problems (Ross, 1984)? The natural contrast is: To what extent do people use strategies such as breadth- first search to solve problems? Related issues include: What characteristics of the current problem trigger a reminded episode? What information about a previous problem gets stored in an episode? What information is utilized from an episode? How is this information used in the current problem? What kind of generalization takes place as a result?

2. ACQUISITION OF SYMPTOM-FAULT RULES: To what extent do people develop symptom-fault rules? Are they generalized from a single episode or from several? If several, how many? What information do they keep? What information do they throw away? Note that other information besides symptoms may constitute the triggering conditions of these rules (e.g., kind of engine, maintenance history, etc.) How are symptom-fault rules related in memory to the episodes that produced them? How are they represented in the mental model? How are they integrated with other knowledge in the mental model? How are they accessed once acquired?

3. ACQUISITION OF QUALITATIVE REASONING RULES: To what extent have rules been acquired that allow accurate simulation with the model? To the extent these rules are present, accurate prediction of how the system will behave should be possible given an input. Of interest is whether predictions about normal functioning are more accurate than predictions about how the engine functions when broken. Both the models of a functioning device and a non-functioning device must be examined here.

4. ACQUISITION OF REASONING STRATEGIES: To what extent do people develop various high-level rules and strategies such as breadth-first search to support troubleshooting? To what extent are reasoning strategies created for troubleshooting the current device, versus being adapted from some other domain or specialized from an abstract strategy? What aspects of the current troubleshooting create or modify these rules? What exactly do these strategies look like for troubleshooting?

5. FURTHER ACQUISITION OF THE MENTAL MODEL: To what extent do people continue developing knowledge of the mental model during troubleshooting? All six kinds of knowledge described in the previous section can be learned. To the extent such learning takes place, what conditions promote it? What processes allow it? Which items are best learned by which processes?

2 MECH: An experimental tool for studying the acquisition of device models and troubleshooting knowledge

In order to systematically study the acquisition of problem solving knowledge from experience, we have had to design an experimental tool that could be used both to present problems to students and to collect data. MECH is both a device simulator and experimental tool, functioning as a presentation and data collection tool and also giving students a way to manipulate the device they are troubleshooting. MECH is an especially important part of the work done on this contract. As an experimental tool built to record the results of experiments in a teaching environment, it has the potential to serve several functions:

- It provides a simulation environment for problem solving, including graphics and help facilities. Thus, with the right knowledge in it, it could be used by students to practice what they have learned without the need for the particular device they have learned about being available.

- It provides an environment for teaching. It has facilities for providing feedback, for providing explanations to students, and for choosing problems to work on. It could therefore be used as a teaching tool.

- It provides an environment for experimentation. It records key strokes and keeps track of latency times. It also allows for different kinds of teaching/learning situations to be set up, thus allowing an experimenter to both explore the learning and problem solving procedures students are using and evaluate the differences between several different teaching strategies.

While MECH's current knowledge allows it to be used to teach automotive troubleshooting, it has been designed with generality in mind. We believe that MECH has much to offer the military in terms of training its personnel about the structure of devices and how to troubleshoot them. Once the necessary data base for a particular device has been developed, MECH can be used for a number of purposes:

1. MECH could be used to train a user quickly about the systems and components in a device, along with the qualitative relations between them.

2. MECH could be used as a teaching device, with instructors using it to demonstrate various troubleshooting techniques (e.g., breadth-first search, symptom-fault rules, qualitative reasoning).

3. MECH could be used to provide students with simulated practice at troubleshooting on their own. Frequent types of repairs could be demonstrated, as well as pitfalls that can occur by misreading symptoms or following faulty decision procedures.

4. MECH could be used as an online manual to remind someone of a device's components and their relations to one another.

5. MECH could be used as an online troubleshooting aid. Because its database contains all possible diagnostic tests and repairs for a device, a user can look up what could possibly be wrong. MECH currently has no information about symptoms and their relations to faults. However, this could be easily integrated into MECH in its current form.

Appendix B explains MECH in detail.

3 Experimental Techniques for Studying the Learning and Troubleshooting of Mental Models

The final task of our psychology team in following up on the year 1 study was to derive experimental techniques for studying subjects learning mental models and reasoning about them and to run some pilot experiments. In particular, we were interested in the role individual experiences play in the acquisition of knowledge about devices and troubleshooting and the evolution over time of the knowledge and reasoning capabilities of our subjects. The techniques we have come up with are sensitive to these things. These techniques are being used in follow-up experimentation in the next phase of the project, supported by a follow-on contract. In this section, we detail the experimental techniques we derived. In Appendix C, we excerpt relevant sections of the Interim Report from the follow-on contract to show where some of these ideas have led.

3.1 Measuring the Acquisition of Mental Models

Earlier we presented six parts of the mental model that are learned: components, hierarchical structure, internal relations, external relations, general physical mechanisms, and multiple mental models. In the context of using MECH, we are not currently able to study the acquisition of general physical mechanisms or multiple mental models (although MECH could be made to handle these with a moderate amount of effort). Consequently, we are only in a position to address the acquisition of components, hierarchical organization, internal relations, and external relations. Arguably, however, these are the most basic aspects of mental models to study.

1. TIMECOURSE ISSUES. There are four timecourse issues that must be considered in assessing acquisition. They are:

   (a) RELATIVE ACCRUAL OF COMPONENTS AND RELATIONS: What kinds of information accrue early in the acquisition of a mental model? What kinds of information come in late? Do components generally precede internal relations? Do external relations generally precede internal relations? Some theories make these predictions. On the other hand, these various types of information may accrue at relatively equal rates. Another issue concerns the heigth of information in the topology of the engine. Are components and relations from higher-level subsystems acquired faster than components and relations from lower-level subsystems?

   (b) RELATIVE ACCRUAL OF ORGANIZATION: How does the organization of a mental model change over time? Are early models relatively unhierarchical, with later models becoming increasingly differentiated according to subsystems? Or are early models primarily organized hierarchically in terms of components, with later models become organized more functionally by internal and external relations?

(c) RELATIVE LOSS OF COMPONENTS AND RELATIONS: After a model has been acquired, which kinds of information are retained the longest and which are forgotten most rapidly? For example, are components better remembered than relations? Are external relations remembered better than internal relations? Are components that are involved in more relations better remembered than components involved in fewer relations? Is information higher in the hierarchy better remembered than information lower in the hierarchy?

(d) RELATIVE LOSS OF ORGANIZATION: To what extent is organizational information lost over time? What kinds of organizational information are best remembered? Is loss of organizational information dependent on hierarchical height? Is organizational information lost more rapidly than component information?

2. EXPERIMENTAL TECHNIQUES. The following experimental techniques can be used to address these questions:

(a) MEASURING MENTAL MODELS AT VARIOUS POINTS IN LEARNING. Subjects could receive multiple tutoring sessions on a physical device, with the same material being presented in each session (i.e., utilizing MECH under experimenter control). At the end of each session, we could assess a subject's current model with various recall and recognition measures. By coding responses with respect to components, internal relations, external relations, organization, and hierarchical height, we can assess how much of each kind of learning occurred. By comparing these measures across sessions, we can see what kinds of learning occur early versus late during acquistion of the model. In other words, we can track the timecourse of learning for these various aspects of the mental model.

Another way to do this study would be to allow subjects to use MECH however they wish to tutor themselves (i.e., utilizing MECH under subject control). We could stop subjects at various points and assess their memory for the different kinds of information. Subjects could return for subsequent sessions of a similar type. By also seeing how subjects search through the tutor to learn, we can also get a sense of the kinds of information they want to see early in learning versus the kind of information they want to see late.

(b) MEASURING MENTAL MODELS AFTER VARYING DELAYS. After subjects have achieved some criterial level of learning, we can test them after varying delays (e.g., immediately, 1 day, 1 week, 2 weeks, 4 weeks, 3 months, 6 months, 1 year). This could be done both between and within subjects. Between-subjects testing would allow the best assessment of what kinds of information are lost at what rate over time. Within- subject testing, in conjunction with between-subject testing, would allow assessing the extent to which testing maintains the mental model in memory. More specifically, subjects could be tested after 1 week, again after 1 month, again after 6 months, and again after 1 year. Of interest would be seeing how their forgetting at each point in time compared to the forgetting of

the comparable between-subjects group. Another useful manipulation would be to run a third condition in which subjects received tutoring instead of testing. More specifically, subjects could be tutored again after 1 week, again after 1 month, again after 6 months, and again after 1 year. Of interest would be seeing how their forgetting at each point in time compared to the forgetting of the comparable within-subjects, testing group. Does subsequent tutoring or testing best maintain a mental model in memory?

(c) MODERATION BY LEARNING CONDITIONS. Actually, assessing the accrual and loss of information in mental models may be moderated by learning conditions. For example, some learning conditions may optimize the learning of components, others may optimize the learning of relations, etc. Several examples of learning conditions are:

- LEARNING COMPONENTS FIRST WITHOUT LEARNING RELATIONS OR ORGANIZATION
- LEARNING INTERNAL RELATIONS WHILE LEARNING COMPONENTS
- LEARNING THE HIERARCHICAL ORGANIZATION OF COMPONENTS WHILE LEARNING COMPONENTS
- LEARNING HIERARCHICAL ORGANIZATION AND EXTERNAL RELATIONS WHILE LEARNING COMPONENTS

The central issue in the above learning methods concerns the proper mix of information to give subjects at various points in learning. Does compartmentalizing information lead to faster or slower learning, and does it lead to better or poorer memory? Or does mixing various types of information optimize learning and memory? If mixing is better, then what are the optimal mixes? Over what timecourse? The basic way to answer these questions is simply to manipulate learning conditions and observe the effects on learning and retention. Another interesting learning mode is:

- LEARNING ABOUT COMPONENTS, RELATIONS, AND ORGANIZATION IN THE CONTEXT OF TROUBLESHOOTING

Subjects may acquire information about a mental model faster if they are trying to troubleshoot it than if their task is simply to memorize the material. This kind of learning may also produce the best retention of mental models. One problem is that the haphazardness of problem solving may make it difficult for subjects to receive systematic and exhaustive coverage of the engine's structure and function. Consequently, a mix of troubleshooting and tutoring may be optimal.

3.2 Measuring the Acquisition of Troubleshooting

Earlier we presented four kinds of learning that could occur during troubleshooting: reminding, symptom-fault rules, qualitative reasoning knowledge, and reasoning strategies. In the

context of using MECH, we are currently able to study all four.

The following nine paradigms may provide various insights into the role that remindings, symptom-fault rules, qualitative reasoning knowledge, and reasoning strategies play in the development of troubleshooting expertise:

1. THE DISTRIBUTION OF STRATEGY TYPES OVER THE DEVELOPMENT OF EXPERTISE
2. FACTORS THAT DETERMINE REMINDINGS
3. FACTORS THAT DETERMINE THE USE OF SYMPTOM-FAULT RULES
4. FACTORS THAT DETERMINE THE USE OF QUALITATIVE REASONING RULES
5. FACTORS THAT DETERMINE THE USE OF REASONING STRATEGIES
6. THE EFFECT OF MENTAL MODELS ON TROUBLESHOOTING
7. THE EFFECT OF TROUBLESHOOTING ON MENTAL MODELS
8. THE ORGANIZATION OF TROUBLESHOOTING EPISODES IN MEMORY
9. THE CONTENT OF EPISODES AND SYMPTOM-FAULT RULES.

Each of these nine paradigms is discussed in turn.

1. THE DISTRIBUTION OF STRATEGY TYPES OVER THE DEVELOPMENT OF EXPERTISE. The basic question is: To what relative extents do subjects use remindings, symptom-fault rules, qualitative reasoning rules, and reasoning strategies over the development of expertise. Do novices initially use reasoning strategies such as breadth-first and depth-first search to find faults? Do they sometimes use qualitative reasoning to map symptoms onto possible faults? After subjects have performed a number of troubleshooting episodes, do they start using remindings to guide search? Once they have been reminded a few times, do they start using symptom-fault rules that are generalizations of remindings? After subjects have acquired symptom-fault rules, what do they do upon receiving a problem that bears no resemblance to a previous problem? Are they more likely now to use qualitative reasoning, or do they fall back on reasoning strategies? Does the likelihood of qualitative reasoning increase or decrease with experience?

   We can study these questions by giving subjects problem sets of 100 problems over several hours, for example, where each problem presents a broken engine with one or more faults. By manipulating the similarity of job characteristics across problems

(i.e., customer, engine type and model, symptoms, maintenance history, previous repairs, customer observations, and faults), we can create conditions that will produce remindings and generalizations. We can identify the use of various strategies using the following techniques:

(a) REASONING STRATEGIES. If subjects are using reasoning strategies such as breadth-first and depth-first search, then we will be able to determine this by the pattern of their keystrokes (which are stored completely during troubleshooting). Subjects using breadth-first search, for example, should test all the highest-level systems first before proceeding to more specific subsystems and terminal components.

(b) QUALITATIVE REASONING RULES. If subjects are using qualitative reasoning rules, then we should see two sorts of patterns in their keystrokes. First, if subjects draw qualitative inferences from symptoms to faults, we should see them go directly to the correct fault without going through something like breadth-first or depth-first search. For example, the symptom "strong smell of gas while engine is running" is connected by various qualitative rules to the choke, throttle, and air intake. If the subject immediately tests these components, we can assume they are pursuing qualitative reasoning of a sort. Second, subjects may use the outcome of a test to direct search, rather than continuing with a meta-rule. For example, if a subject discovers that no fuel is reaching the cylinder and that the intake valve is not broken, then the subject may reason that something in the carburetor must be clogged or broken.

(c) REMINDINGS. If subjects are using remindings, then upon receiving a job that shares characteristics with one previous job, they should immediately test the component(s) at fault in the previous job. They should not use a meta-rule or qualitative reasoning to determine search.

(d) SYMPTOM-FAULT RULES. If subjects are using symptom-fault rules, then upon receiving a job that shares characteristics with two or more previous jobs, they should immediately test the component(s) at fault in the previous job. It is essential to note that subjects could be using remindings at this point. In general, it is difficult if not impossible to discriminate exemplar from generalization models, empirically speaking. There may be a fundamental indeterminacy problem here that can not be resolved, much like the indeterminacy problems for serial versus parallel processing and imaginal representations versus propositional representations. Nevertheless, we may discover that there are ways to differentiate these accounts, and we will attempt to do so. Otherwise, we will probably make a theoretical assumption that subjects who show learning after receiving two or more problems of a particular type have extracted a symptom-fault rule. Issues concerning the induction of symptom-fault rules are addressed in a later section.

In summary, we will use patterns in subjects' keystroke files, in conjunction with our

knowledge of the problems they have received, to assess their strategies in locating faults. Again, our primary interest will be to see the relative extent to which these strategies are used over the development of expertise.

2. FACTORS THAT DETERMINE REMINDINGS. Basic questions include: To what extent must the current job overlap in features with a previous job for the previous job to be reminded? What kinds of features produce the most frequent remindings, holding amount of overlap constant? Are remindings more likely to occur early in learning rather than late? To what extent must a previous job have occurred recently for it to be reminded? Finally, imagine that a previous job (the "target job') is similar to the current job on several features. Further imagine that we vary the extent to which other previous jobs are similar to target job on features different from those shared by the target and current jobs. Does this decrease or increase the probability of the current job reminding the target job? In other words. if the target job is part of a cluster that may have been generalized, how does this affect accessibility of the target?

   We can study all of these questions in MECH by constructing pairs of problems that overlap on certain features. We can manipulate the number of shared features, the type of shared features, the number of intervening problems, and whether the pair occurs early or late in learning (holding the number of intervening problems constant). We can also manipulate the similarity of previous problems to the target. In all cases, we can assume reminding has occurred if a subject first tests the same component that was at fault in the target problem.

3. FACTORS THAT DETERMINE THE USE OF SYMPTOM-FAULT RULES. Basic questions include: To what extent must two or more jobs overlap in features for a symptom-fault rule to be constructed? What kinds of features are most likely to produce a symptom-fault rule, holding amount of overlap constant? How does the probability of forming a symptom-fault rule increase with the number of similar jobs? Are symptom-fault rules more likely to develop late in learning rather than early in learning? Are massed or distributed episodes more likely to produce symptom-fault rules? When a symtpom-fault rule is formed, what information is extracted? Only information that can be related to the fault by qualitative reasoning? Or irrelevant information as well? If irrelevant information is included, is it just as likely to trigger the symptom-fault rule as is relevant information? Finally, how are symptom-fault rules related in memory to the episodes that produced them? Are they clustered together such that activating the symptom-fault rules also activates the episodes? Or are they stored separately?

   We can study all of these questions in MECH by constructing sets of problems that overlap on certain features. We can manipulate the number of shared features, the type of shared features, and the number of problems sharing features. We can manipulate whether similar jobs are massed or distributed and whether they occur early or late

in learning. We can see whether the conditions for a symptom-fault rule contain irrelevant as well as relevant information by seeing if later problems that only contain the irrelevant information fire the rule. In all cases, we can assume that a symptom-fault rule has been formed if a subject first tests the same component that was at fault in the previous problems that produced the rule. Again it is important to note the difficulty of disciminating pure exemplar accounts from generalization accounts. Whether a symptom-fault rule is stored with its episodes is addressed in the paradigm that addresses the organization of episodes, discussed below.

4. FACTORS THAT DETERMINE THE USE OF QUALITATIVE REASONING RULES. Basic questions include: Does prior training on the structure and function of a physical device transfer to qualitative reasoning during troubleshooting? If a subject first learns internal and external relations from a tutor, does this later facilitate reasoning about how a symptom might be produced by a faulty component? Or about how a faulty component might affect another component? Perhaps subjects really only learn to reason qualitatively in the process of troubleshooting. If so, then what types of troubleshooting experience best promote the acquistion of qualitative reasoning rules? To the extent problems form predictable clusters and produce predictive symptom-fault rules, do subjects not learn to reason qualitatively? To the extent problems don't have much predictive structure at all, do subjects primarily use meta-rules and forego qualitative reasoning? Does qualitative reasoning primarily develop when subjects are faced with relatively novel problems, where knowing qualitative relations between components can facilitate search? If so, do these problems promote better qualitative reasoning skills than learning about qualitative relations from a tutor?

   We can study these problems in MECH in a couple of ways. First, we can vary the kind of tutoring a subject receives before troubleshooting to see if tutoring affects the ability to reason qualitatively. If tutoring can promote this skill, then we should see benefits from some types of tutoring but not others. Second, we can study qualitative reasoning by controlling the composition of the problem set. We can manipulate the amount and type of predictive structure in the problem set to see if these factors determine how well subjects reason qualitatively.

   Measuring the ability to reason qualitatively can be done in two ways. First, we can ask subjects specific questions about the relations between two components. For example, "If the magneto is broken, what other components might not function properly?" Or, "If the magneto is not working properly but is not broken, what other components might be broken, thereby causing the magnetoto malfunction?" Second, we can observe qualitative reasoning indirectly by looking at how subjects search for faults. On some problems, symptoms may be qualitatively related to the faults. If subjects are good at qualitative reasoning, they should make the connection and find the fault quickly. On other problems, we can direct subjects to a component that is not broken but that is not working properly. If subjects are good at qualitative reasoning, they should converge quickly on the faulty component that is making the unbroken component

perform improperly.

5. FACTORS THAT DETERMINE THE USE OF REASONING STRATEGIES. Basic questions include: Do subjects generally prefer breadth-first or depth-first search? What conditions promote these preferences? What other general rules do subjects develop to direct problem solving?

   To see whether subjects prefer breadth-first to depth-first search, we will in some cases present them with problem sets that have no predictive structure and see what they do. To see whether different conditions promote these two types of search, we will simply note situations where a particular type of search is preferred. At this point, it is not clear what these situations might be. As far as other general rules, we again don't have any specific hypotheses and will simply be on the look out for systematic patterns that suggest the use of meta-rules.

6. THE EFFECT OF MENTAL MODELS ON TROUBLESHOOTING. The basic question here is: What effect does prior tutoring on a mental model have on troubleshooting? Are subjects any better at troubleshooting after having learned the structure and function of the device to be repaired? If so, then are particular types of tutoring better than others at promoting troubleshooting skill? Does focusing on hierarchical structure during tutoring encourage breadth-first and depth-first search during troubleshooting? Does exposing subjects to all the possible tests and repairs during tutoring lead to better troubleshooting later? If so, does presenting tests and repairs work best when they are presented breadth- first, depth-first, or following paths of qualitative reasoning?

   To assess this question, we will tutor subjects in various ways described in Section 5 and see what kinds of tutoring produce the best troubleshooting. We will also include a condition with no troubleshooting to see if these subjects do as well as tutored subjects. The measure of troubleshooting ability will simply be how quickly subjects find faults (i.e., how closely actual costs approximate ideal costs in the payoff structure). We will also look more specifically at the abilities to reason qualitatively, to use meta-rules, and to construct symptom-fault rules, all of which may be affected by various kinds of tutoring.

7. THE EFFECT OF TROUBLESHOOTING ON MENTAL MODELS. The basic question is: What effect does troubleshooting have on learning mental models? Do people learn mental models better in the context of troubleshooting than in a tutoring context? If troubleshooting primarily serves to increase the quality of a mental model, what kinds of changes does it produce?

   We can explore these questions by assessing people's mental models after troubleshooting, using all the same measures described earlier in Section 5 (e.g., knowledge of components, internal relations, external relations, hierarchical organization). If troubleshooting alone produces better learning than tutoring alone (given a constant amount of time spent), then troubleshooting subjects should score higher on these measures

than tutoring subjects. To see how troubleshooting changes already established models, we can first tutor subjects on a mental model, then have them perform troubleshooting, and then assess their mental model. We can compare these subjects to others who were tutored but who did not perform troubleshooting. We can then see what kinds of differences exist between the mental models of these two groups on our variety of memory measures.

8. THE ORGANIZATION OF TROUBLESHOOTING EPISODES IN MEMORY. The basic question is: How are episodes integrated with symptom-fault rules and qualitative reasoning rules? If subjects store episodes with the rules that identified their faults, then subjects should later cluster episodes that share a common rule. For example, if several episodes involved the same qualitative reasoning rule, then these episodes should be recalled together. If several episodes involved the same symptom-fault rule, then these episodes should be recalled together. Another possibility is that subjects store episodes according to the topology of the model, with each episode being associated with its fault(s) or with every component that was tested.

   We can assess this issue by asking subjects to recall all previous jobs at the end of troubleshooting. To the extent that subjects have organized exemplars according to particular organizational principles, we should see jobs clustered in those ways.

9. THE CONTENT OF EPISODES AND SYMPTOM-FAULT RULES. The basic question is: What information is recorded in memory for episodes and symptom fault rules. Are memories of episodes biased toward information relevant to finding the fault? Or is irrelevant information remembered well, too? Do symptom-fault rules contain only predictive information? Or do subjects also generalize over irrelevant information that is not predictive?

   To assess these questions, we can ask subjects to recall information about previous jobs and about the conditions of symptom-fault rules. For episodes, we can give subjects enough information to identify a job and then ask them to recall the remaining details. We can further probe their memories by specifically asking them to recall the type of engine, the maintenence history, etc. For symptom-fault rules, we can give subjects a fault that occurred in more than one job and ask them to provide characteristics shared by jobs having that fault. Again we can probe subjects by specifically asking them to recall the type of engine, the maintenence history, etc. Using these recall techniques, we can assess the information stored with episodes and symptom-fault rules.

4 Exploring learning processes through AI modeling

Our other major thrust has been to build computational models of some of the processes we saw our subjects using to find out more about how these processes might work. Our intention in building systems has been to find out more about what knowledge is needed and

what knowledge is helpful in learning how to solve problems and under what circumstances it is easy to acquire that knowledge from experience. Modeling reasoning processes on the computer gives a means of investigating how processes work in ways that are not possible using people. It also gives us a way to state concretely the steps involved in carrying out the reasoning. In essense, we have been building a model of the ideal novice problem solver. Feedback from this model building process serves two purposes: It informs subsequent systematic experimentation with people, showing where effort should be placed in experimentation, and it might ultimately provide us with guidelines for developing teaching strategies and systems that teach.

4.1 The Active Learner

Work on CELIA has been aimed at modelling the reasoning and learning processes of a student learning by watching and listening to a teacher explain particular troubleshooting cases. A novel feature of this model is its commitment to **case-based reasoning** as a natural reasoning process of the learner. Using case-based reasoning, a reasoner solves new problems by making reference to previous situations similar to the current one and adapting the solutions to the old problem to the new situation.

The model we have developed based on the protocols collected in Year 1 is a model of the student as an active intentional learner. As this active learner watches and listens to a teacher performing some troubleshooting task, he/she/it predicts what the teacher will do or say next. When predictions don't match what the teacher says or does, the reasoner attempts to explain the discrepancy and learns from it. Case-based reasoning's major role is in aiding the prediction process. Predictions that the student makes often come from previous troubleshooting cases remembered in the course of reasoning.

Descrepancies play a large role in the model. The program compares its predictions with what the teacher does and when they differ, sets up explicit learning goals. To do that, it characterizes the reasoning failure and figures out what it would have needed to know to get it right. This results in learning goals that the reasoner then attempts to pursue through appropriate plans.

The framework is a general approach that can be used to unify learning of several different kinds of concepts learned during troubleshooting:

- learning an unknown goal (e.g., the reasoner didn't know that a complaint needs to be verified)
- learning feature salience (i.e., under what conditions certain features are important to attend to)
- Learning By Understanding Explanations (i.e., filling in gaps in one's causal knowledge by integrating a teacher's explanations with what is already known)

In addition, it is a framework that can support several other learning processes:

- Adjusting selection of predictions from possibilities
- Filling a gap when faced with reference to unfamiliar information
- Filling a gap when lacking an answer to an instructor's question
- Learning by asking focused questions.
- Integrating an instructor's hint into domain knowledge
- Directed reading

Articles that further describe CELIA and the model it implements can be found in Appendices D and E.

4.2 Learning by Understanding Explanations

Within this framework for intentional learning, one area we have studied extensively is learning by understanding an instructor's explanations (LBUE), one of the learning processes we observed in student car mechanics. In LBUE, the motivated student searches for explanations of why the instructor performed a particular action or proposed a particular solution. The instructor's role is to help the student complete explanations when the student's understanding of the domain is somehow deficient. The instructor does this by revealing some hidden but necessary reasoning of which the student might not be aware. The student then uses this information to help complete his understanding of the example. A more complete understanding permits more learning. LBUE tells us much about both learning and instruction. Papers in Appendix D explain LBUE in detail.

Figure 1 summarizes the general LBUE process. Essentially, the instructor presents the problem, an appropriate action or solution, and any hidden reasoning that the instructor considers useful to the student. The student uses the supplied hidden reasoning to understand or explain why the instructor's action or solution is appropriate. In general, a student or learning system applying this algorithm has a background domain model that is not necessarily complete or consistent and a set of procedural or declarative rules that help direct the search for explanations in the given domain.

---

1. For each example, the *instructor* states and refines the problem description and goals.
2. The *student* attempts to generate an appropriate action or solution for the problem.
3. If necessary, the *instructor* generates a correct action or solution.

4. The *student* then attempts to explain this action by reasoning in both directions, from the problem description and from the solution.
    - small gaps are filled by some form of plausible inference, and complete explanations can be generalized.
5. If necessary, the *instructor* provides a partial explanation revealing some subset of the hidden results or processes.
6. The *student* then incorporates any unknown information about these hidden results or processes and attempts to complete the explanation, repeating step 4.

Figure 1: General LBUE Algorithm.

LBUE predicts many types of student learning. First, the student has the opportunity to learn about intermediate results or hidden processes that would assist in completing an explanation for the current example and for future examples. This permits students to fill in gaps in their domain models. Second, the students can learn more efficient organizations of their knowledge by using Explanation-Based Learning techniques to generalize the conditions under which the given action or solution was appropriate. LBUE can potentially produce a system more efficient than a standard inductive system and can learn truly new knowledge unlike standard EBL.

In troubleshooting, as in the general model, learning is triggered when a complete explanation cannot be found for an instructor's proposed solutions. In diagnosis, these solutions are statements of hypotheses. The instructor presents an automobile that is malfunctioning, and he generates possible hypotheses of the underlying cause. The student or learning system then attempts to explain why the hypothesis is appropriate. One way to explain hypotheses is to attempt *causal chaining.*

Often a student will not have enough information to explain the hypothesis. For example, a student may not know what a cracked distributor cap can cause. In general, the student's model of a domain cannot be expected to be complete and consistent. When important information is missing, the instructor's explanation can be useful. LBUE permits three types of learning in any domain, including diagnosis.

- If the instructor's partial explanation represents a new causal relation, such as X causes Y, the partial explanation can be added directly to the causal domain model with high credibility. For example, when the instructor says that a cracked distributor cap can allow moisture to collect inside the distributor, that information can be added reliably to the student's causal domain model.
- The instructor's partial explanation may allow bridging a gap to complete a *causal chain*, thus enabling the student to add a collapsed *causal chain* as a new association in the causal knowledge, as in EBL. In our example, once the student has been told the partial explanation, she can then realize that moisture in the distributor cap could

interfere with the electricity, causing a lack of electricity reaching the spark plug, which, in turn, would prevent the spark plug from firing. A complete explanation has been formed.

- A final type of learning can occur when the explanation does not complete a chain. In this situation, the student may still infer a causal relationship that fills a gap in the explanation. For example, if the student did not know that moisture in the distributor cap could interfere with the electricity, then there would still be a gap in the student's explanation. This might be able to be filled, depending on available general knowledge.

LBUE, as applied to learning automobile diagnosis from examples, uses a very straightforward learning algorithm. An outline of this algorithm as implemented in EDSEL1 and EDSEL2 is shown in Figure 2. This algorithm describes the details of the general LBUE process (Figure 1) from a diagnostic student's point of view.

---

1. When the symptom is presented, attempt to causally chain backward toward possible root causes, trying to determine what could cause the symptom.
2. As each hypothesis is presented, attempt to causally chain forward toward possible effects, trying to determine what the hypothesized state could cause.
3. If the symptom causal chain meets a hypothesis causal chain, then the EBL generalization is added to the causal domain model.
4. If the causal chains do not meet,
   (a) Causal chain backwards from the instructor's partial causal explanation toward the hypotheses.
   (b) Chain forward from that partial explanation toward the symptom.
   (c) If both directions can be linked, then the most general relationship *(cause hypothesis symptom)* can be learned.
   (d) If a gap still exists, then try to infer the missing causal links.
5. If necessary, add the partial causal explanation to the causal domain model.

Figure 2: LBUE Algorithm Specialized for Diagnosis.

After this learning, the causal domain model (mental model) is more capable of completing an explanation, and diagnosis is more efficient and more powerful for the same or similar problems. In general, less knowledgeable students will have bigger gaps to fill, such that many plausible inferences are possible. Large inferential leaps can be refined and corrected by further diagnostic episodes. The LBUE process is able to augment and correct a causal model, and, as with people, what is learned is strongly influenced by what is already known.

4.3 Case-Based Reasoning

Of particular import in this model of a student is case-based reasoning. In case-based reasoning, inferences are made based on particular experiences rather than based on general knowledge or first principles. Old cases that are remembered in the course of reasoning serve

several purposes: proposing shortcuts to solutions and warning the reasoner of the potential for problems. Case-based reasoning allows a reasoner to make inferences by association rather than by tracing through a complex causal model. Of particular interest in case-based reasoning is how to index into a case memory to choose appropriate cases. Our work has led us to hypothesize five ways in which troubleshooting cases are indexed in memory.

- Symptoms, along with other features, are used to recall cases that can predict faults (hypotheses)
- Hypotheses, along with other features, are used to recall cases that can suggest tests to be done.
- Tests and their results, along with other features, are used to recall cases that can suggest hypotheses or interpretations.
- Faults, along with other features, are used to recall cases that can suggest repairs.
- Symptoms, along with other features, are used to recall cases that can suggest repairs when faults cannot be tracked down.

"Other features", in each of the descriptions above, are particular to each individual case and are the features that in addition to the one of the type specified are predictive in the right way. Some symptoms, along with a make and year of car, for example, are predictive of a particular fault. The same symptom, along with some other feature, such as who fixed the car last, might predict a different fault.

Also of interest in case-based reasoning is the representation of cases. We have discovered that a representation that breaks cases into case pieces is most advantageous. Case pieces represent hypothesis-test-result triples and sequences, and are stored in memory indexed off of the engine part or system that is hypothesized to be broken. Case pieces include any information needed to explain why the hypothesis was made, the hypothesis itself, the test that was done, the result, and any information that can help explain the result. Pieces of the same case point to each other in such a way that a whole case can be reconstructed from the pieces. Cases are used for a variety of problem solving tasks. The description of index types above gives an idea of the inferences they can help with.

Appendix E holds papers about case representation.

4.4 Memory Organization and Representation

If case-based reasoning and explicit reasoning about a causal model are both used in solving problems, then another issue of importance is the integration of cases and knowledge about devices and cases. We have spent considerable time on developing representations of mechanical devices and organizing those representations in a memory. There are several principles we base our representations and organization on:

- Memory must represent both general knowledge about mental models and specific knowledge about particular cases. Both should be in the same form and accessible by the same retrieval algorithms.
- Memory organization must integrate general knowledge and specifics of cases in a natural way. That is, if a case was used to build or elaborate some part of the model, it should be associated with that part of the model. Conversely, if a case illustrates or specializes some part of the model, it should be associated with that part of the model.
- Models of normal function and models of malfunction should be appropriately integrated. We do it by associating both with the part or system they go with.
- Since people can make inferences based on either the general model or specific cases, inference mechanisms should be able to work on both.
- The mental model must represent normal function, malfunction, physical struccture, flow, and causality. People use all of these in troubleshooting.
- In representing cases (experience), the order in which hypotheses were made is important and should be explicitly represented.
- On the other hand, access to cases suggests that they should be represented and stored in pieces with connection between the pieces and separate indexes to each piece. Pieces seem to be associated with hypotheses made during troubleshooting.

These principles result in memory orgnaization that is hierarchical in several ways. First, a partonomic hierarchy reflects the system/subsystem aspects of the model. Second, a generalization/specialization hierarchy reflects the progression from general knowledge (a carborator has the following parts) to more specific knowledge (carborators in Nissan Stanzas have an additional part called an x with the following features) to very specific knowledge (the carborator in Kolodner's Nissan Stanza won't hold its idle setting. As a result, gum has been put around the idle control to hold it in place.) Generalization/specialization hierarchies are used to represent both general knowledge and experience with both normally functioning and malfunctioning cars.

Each part represented in memory has topological relations specified (near what, connected to what), input and output, and function. Each has a set of inferences describing it in normal functioning mode and a set describing its various malfunctions (how to recognize them, results of malfunction, how to test for it, how to fix it). These form the normal function and malfunction model of the part.

Cases are integrated with general knowledge by breaking them into pieces based on the hypotheses made about malfunctions. Each hypothesis is associated with a malfunctioning part of the car, and each case piece lives in a generalization/specialization hierarchy associated with the part its hypothesis is about. Case pieces hold connections to other case pieces they come before or after or predict.

We also need to designate what access functions look like. How are these different kinds of knowledge retrieved? We want retrieval processes to model people's processes. This is the way we can predict what learning situations are effective, what people will recall in any situation, and the kinds of external memory organizations that will be effective. Several principles for memory retrieval are:

- More specific information should take preference over less specific upon retrieval.
- The same procedure should recall both specific and general knowledge, hence both should be organized and indexed the same way.
- Retrieval should not slow down with the addition of information to the memory.
- Update processes should access memory as retrieval processes do, putting new information in correct places.

We have worked on two approaches to this problem – one in the context of CELIA, the other in a separate memory model. CELIA's model is presented in papers in Appendix E. CORA is the separate memory model, and it begins to implement a model with the constraints above. Appendix F holds papers with details about CORA. CORA is novel in the realm of AI memory models in being based on conditional probabilities. Cases in CORA are stored overlapped with other cases in such a way as to allow case reconstruction. This overlapping storage is achieved by maintaining conditional probabilities between case pieces. In a sense, the memory does not store cases individually, but rather stores the information necessary to rebuild the case given a partial description of its features. Cases are retrieved from CORA's memory by incrementally adding the most probable correlates to the retrieval cue. CORA is capable of rapidly retrieving the best-match case or cases from a very large case store. As well, CORA automatically generates generalizations by similarity-based learning, and upon retrieval of a case that is a lot like several others, it will recall the generalizations it made, returning a composite rather than an exact case. CORA uses its conditional probabilities to index on unusual features or attributes.

The motivation for CORA was based on results from the study of human memory. Human memory for categories and language has often been described as representing correlational feature structure (Rosch, 1978; Medin & Schaffer, 1983; Billman, Heit, & Dorfman, 1987), and this structure may provide a basis for the initial learning of these types of information (Billman, Heit, & Dorfman, 1987). Anderson (1988) further argues that a Bayesian analysis is the most appropriate way to describe and explain memory retrieval, skill learning, and the representation of category structure. The application of these ideas to a machine memory constitutes an important contribution to artificial intelligence in the fields of conceptual clustering and representation, knowledge acquisition in general, and knowledge access from large databases. Conditional probabilities have been used in several machine learning projects, most notably those of Schlimmer (1987) and Fisher (1987).

Appendix A

GIT-ICS-89/02
Problem Solving and Learning in a Natural Task Domain

Juliana S. Lancaster*
Janet L. Kolodner

School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332

*BDM Corporation
1300 N. 17th St., Suite 950
Arlington, VA 22209

November, 1988

This research was supported in part by the Army Research Institute for the Behavioral Sciences under Contract No. MDA-903-86-C-173. Correspondence should be addressed to Janet Kolodner, School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA 30332.

Problem Solving and Learning in a Natural Task Domain*

Juliana Lancaster
BDM Corporation
1300 N. 17th St., Suite 950
Arlington, VA 22209

Janet L. Kolodner
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, GA 30332

November 16, 1988

Abstract

Problem solving is known to vary in some predictable ways as a function of experience. This work has sought to uncover the particular processes by which problem solving knowledge is acquired from experience and integrated with already-known knowledge to be used in later problem solving. Protocol collection and analysis have allowed us to uncover some of the problem solving and learning processes used by novice and more experienced mechanics, as well as the knowledge they have available. Work on representing that knowledge in a way that can support the problem solving processes we observed allowed us to begin extending our understanding of these processes. This report describes a series of protocol experiments and their analysis, and some initial attempts at representing the knowledge used by the subject mechanics.

*This research was supported in part by the Army Research Institute for the Behavioral Sciences under Contract No. MDA-903-86-C-173. Thanks to Phyllis Koton, Joel Martin, and Michael Redmond for commenting on earlier drafts of the paper.

1 Introduction

In many domains, it is obvious that expertise is a function more of experience than of formal training or success in passing examinations. Nevertheless, research on problem solving and on the nature of expertise has often examined variables unrelated to the individual experiences of the learner, or even the course of the learner's study. In earlier work on problem solving for example, the variable of interest was most often the number of steps required to complete a problem or the number of correctly solved problems within a set. As research turned to more and more complex domains, the differences between novices and experts began to reflect interesting qualitative shifts in the nature of problem solving and in the knowledge used in solving problems. As a consequence, researchers began to concern themselves with characterizing those differences and shifts. Much has been learned over the past twenty years about differences between novices and experts (e.g., Bhaskar & Simon, 1977; Chi, Glaser & Rees, 1982; Glaser, 1985), but to date little has been said about the role(s) of specific experiences in the process of becoming an expert.

This work has been conducted to enhance our understanding of the roles that specific experiences and interactions can have on a novice's changing knowledge and problem solving behavior. In order to consider this issue, we first examined three characteristics of the knowledge held by problem solvers: availablity of knowledge before solving any problems; available knowledge for the solution or later problems; and knowledge organization. We then explored the learning processes used at different levels of expertise in order to identify the kinds of learning aids that might be provided for students, and identified the circumstances under which particular experiences are memorable.

The task domain selected was auto mechanics. This domain is particularly appropriate for several reasons. It is highly complex in that the automobile engine consists of several interacting systems that combine to produce the desired effect (motion of the car). Failures in one component or system may be manifested purely within that system, or may have compound effects that reveal themselves in a malfunction within another system. These observable symptoms are used to locate the failure, but are not generally interpretable by the amateur. The domain is knowledge-rich and the depth of knowledge and ability to use it are both important in making a good diagnosis. Schools teach about cars in general, but since there are so many different kinds of cars, each having its own peculiarities, textbooks and schools cannot teach everything. Diagnosing a particular car may depend as much on the age and style of engine as on the given symptoms. In addition, the problem description given by the owner to the mechanic is frequently incomplete

and/or inaccurate. Thus experience with many different types of cars and problems is essential to gaining expertise.

2 Methods

The results reported here are based on an analysis of two sets of problem solving protocols collected in separate studies. In both, student mechanics were observed at weekly intervals while diagnosing car failures. Each failure was introduced deliberately into the car and each problem was caused by only one failed part. Think-aloud protocols were collected while the students worked and were transcribed and coded for later analysis. During the first study, the instructor demonstrated the correct or optimum troubleshooting sequence for diagnosis of each failure after all subjects were finished. Thus, each student had an opportunity to obtain feedback on his performance and an explanation of the car's problem whether or not he had diagnosed it correctly. In the second study, the problems were specifically sequenced so that we could see the effects of earlier learning on later problem solving. In this set, the instructor's demonstrations were omitted.

Analysis of the data derived from these observations focused on the knowledge and strategies used by students at different levels of training, how their knowledge was organized, and how their knowledge and strategies changed with experience. We expected that the more experienced student would solve more problems and would give evidence of having a more organized knowledge base than the less experienced students. In addition, we expected that individuals would show evidence over the series of problems of acquiring new diagnostic skills and new knowledge and connections within their knowledge. Protocols from the second set were also analyzed for evidence of case-based reasoning.

2.1 Subjects

The subjects were student volunteers at a post secondary technical school. The technical program is a two-year, eight-quarter program. During much of the second year, the students work in a shop setting within the school. Cars belonging to school personnel and friends of the students and instructors are diagnosed and repaired by students. In addition, the school owns several cars that can be used in teaching students to teach about specific problems.

Each of the student volunteers was at a different point in the program. The novice student in Set 1 was in his first quarter of the program and had no prior training or experience. Intermediate students were at the beginning of the second

year in the program and had no relevant work experience. The advanced students were near the end of the second year. One advanced student, who participated in both sets, also held a part-time job as a mechanic outside of school. Each student worked on at least four of six problems in the problem set he participated in.

2.2 Procedure

The procedure used for protocol collection was the same for both problem sets. In each session, the student was led to the car and, with the experimenter posing as a customer, told that the car was exhibiting a particular symptom. The student was then allowed to perform any tests desired on the car and its engine, with the exception of a driving road test, prohibited primarily by the symptoms presented by the car. The student was instructed to think aloud as he worked to find the failed component in the car. His comments were tape recorded by the experimenter, who also served as an assistant to the student when necessary. Subjects were observed once a week while diagnosing an actual problem in a car.

The problems used were selected by an instructor in the program in consultation with the experimenter. The problems used in Set 1 and the information given as the customer's complaint are described in Table 1. Comparable information for the problems in Set 2 is given in Table 2. Each fault was introduced into a car by the instructor or by a student not in the study under the direction of the instructor. The cars used were all owned by the school with one exception: For one problem in the first set (1B), we used a car brought in by a school official that had symptoms we had been presented to the students in the previous week. In every case, a single complaint was given and a single fault could be traced to account for the complaint. Students were told to track down the fault, but not to fix it unless repair was necessary to confirm the diagnosis.

2.3 Coding

Both sets of protocols were coded the same way. After all protocols were transcribed, each statement was coded into one of six categories, shown in Table 3. Statements coded as *hypotheses* were those in which a specific system or component was first named as a possible source of the failure or in which the system or component was accepted or rejected as the source of the failure. Hypotheses were numbered in order of appearance and, each time one was mentioned, its status was noted. Its status could be *open, accepted, confirmed, or rejected. Rules* were statements giving known, constant information about an engine or about the process of diagnosis. Statements coded as *information gathering* were generally descriptions

Table 1: Problems in Set 1

PROBLEM #	CAUSE (fault)	SYMPTOM (complaint)
1A	blockage in gas line	cranks but will not start
1B	bad cell in battery – will not hold charge	cranks slowly when starting
1C	bad connection behind fuse panel and fuel pump fuse	cranks but will not start
1D	loose ground wires from Electronic Control Module (computer)	cranks but will not start
1E	open tack circuit	cranks but will not start
1F	poorly adjusted timing	detonation on acceleration

Table 2: Problems in Set 2

PROBLEM #	CAUSE	SYMPTOM	# SOLVING/ATTEMPTING
2A	bad plug wire	will not start	3/3
2B	closed plug gap	will not start	4/4
2C	rocker arm gone	will not start	2/4
2D	restricted fuel	engine surges	0/2
2E	restricted fuel	engine surges	2/4
2F	vacuum line hole	engine surges	0/3

Table 3: Coding Categories for Protocols

CATEGORY	ADDITIONAL SPECIFICATIONS	EXAMPLES
Hypotheses	Number and Status	Could be starved for gas (N-P1)
		It could be, could be the starter (N-P2)
Rules	Topic(Failure, normal functioning, or troubleshooting)	Fuel Pump should come on for 3 seconds (I-P4)
		First of all, I have to locate the connector to the back of the fuel pump (A-P3)
Information Gathering	Source of information obtained	Before I look in the book, I'm going to check the fuse (A-P3)
Observation	Topic (hypothesis(number) or complaint)	What we don't have is fuel to the throttle body (A-P3)
		I don't believe I hear it running (A-P3)
Restatements	Topic (complaint or summary of observations)	to rephrase that-the throttle body is not injecting fuel (A-P3)

of the actions being taken by the subject at the time. Such actions could elicit or obtain information from the customer, from a book, or via a procedure or test applied to the engine. *Observations* were statements giving the information obtained from the action taken. ***Restatements*** were repetitions of previously stated or collected information rather than new information. Each statement falling into one of the last three categories was identified with a specific hypothesis by its number if possible. All other statements were uncodable and were marked as such.

2.4 Analysis

The coded protocols were then analyzed for several different things: what subjects at different levels of expertise knew and how their knowledge was organized; how their knowledge changed (i.e., what they learned) as they progressed; and the processes by which they learned.

To analyze what each student knew and how his knowledge was organized, we summarized the coded data for each problem in two ways. First, each coding from a protocol was listed down a vertical column. From this information, we counted

the number of different hypotheses used by a student for a given problem and the number of correct diagnoses made by each student. These columns were then placed side by side, and any use of information novel to the student in one problem that was used easily in a later problem was marked by connecting the two codings. This information was useful as a rough measure of whether or not the students were learning from the problems. In the second summarization, each hypothesis, action, or conclusion of a student on a given problem was noted in a short verbal form (e.g., HYP: Bad fuel pump; check vacuum lines for leaks; check carb for dirt or gas; manipulate throttle by hand). Alongside each, we noted the justification or rule used to support the action, hypothesis or conclusion. When all protocols had been summarized, we composed models of the engine and of diagnostic possibilities and procedures held by each student by collecting all statements about the normal state of the engine into the engine model, all hypotheses into the symptom-fault model, and all procedures into a procedure model. Using the information, we compared both the amount and nature of the models held by each student. The results of these comparisons are discussed in Section 4.

In order to analyze the learning that each of our subjects accomplished, and the methods by which they learned, we chose two sequential protocols for each student after protocols were coded. The two protocols were selected to have the same initial symptom, but different failures. For each of these protocols, a description of the knowledge used and sought by the student during diagnosis was compiled. The descriptions of the two protocols were then compared and differences were noted. Of particular interest were occasions in which the student appeared to use knowledge in solving the second problem that he did not have in solving the first problem. Changes in the knowledge used showed us what students learned between solving the two cases. We then examined the protocols and the instructor's explanation of how to solve the first problem to see if we could identify how these items were learned. Sometimes this was obvious – a student had asked a question previously or the instructor had presented the relevant material in a previous explanation. Sometimes it was not obvious – what was learned could have been presented in class between the two exercises. We focused on those learned items for which we could identify the situation in which the new concepts were learned.

3 General capabilities of novices and experts

As expected, the ability of the students to correctly diagnose the problems changed substantially between the novice level and the intermediate and advanced levels. The diagnoses given by each subject and the number of hypotheses considered are shown in Table 4 for the first set of problems. The novice correctly diagnosed only

Table 4: Final Diagnoses and Number of Hypotheses Considered by Each Subject

PROBLEM	NOVICE	INTERMEDIATE	ADVANCED
1A	not getting fuel(4)	clogged fuel line(3)	——
1B	dead battery cell(5)	starter(4)	dead battery cell(6)
1C	——	fuel pump relay(5)	fuel pump fuse(5)
1D	fuel pump(3)	no diagnosis(6)	injector solenoid(9)
1E	no diagnosis(0)	open tach circuit(9)	——
1F	——	bad timing(10)	bad timing(2)

one of four problems attempted, while the intermediate student correctly diagnosed three of six and the advanced student three of four. In addition, the number of hypotheses considered increased with expertise. The novice generated a mean of 3.0 hypotheses per problem and the intermediate and advanced students generated 6.8 and 5.0 hypotheses per problem respectively.

4 What was known; what was learned

In general, the diagnostic behavior we saw was similar to that reported by other researchers (Hunt, 1981; Rasmussen, 1978; 1979; Rasmussen & Jensen, 1974). Students generated one or more possible hypotheses for the failure immediately after observing the symptom(s). These hypotheses were then tested in a fairly systematic (albeit sometimes idiosyncratic) way either by observation of the inputs to and outputs from specific components and systems or by performance of specific diagnostic tests. In successful cases, a single diagnosis ultimately was given, accompanied by an explanation of how or why that failure would generate the observed symptom(s).

4.1 Knowledge Structures and Organization

We interpret this process as being indicative of an interaction between two types of knowledge structures. The first, a *causal model* of the car's engine, contains knowledge about individual components and their inputs, outputs, and normal behavior; relates components within a system to one another; and describes the relationships and connections between systems. It is used to evaluate hypotheses in light of the evidence obtained from the failed engine and to lead the mechanic through the engine to the source of the problem in a systematic way. The causal model is generally quite large, and the second type of knowledge structure, *symptom-fault sets*, is used

to index into the causal model at appropriate places. Symptom-fault sets represent the relationships between particular symptoms or sets of symptoms and failures. For example, given the symptom "the car cranks but will not start", the symptom-fault sets will identify three systems as possible locations for the failure: the fuel system, the air intake system, and the ignition system. Within each of these systems, additional symptom-fault sets will identify individual components that may cause the symptom(s). For the fuel system, these would be a failed fuel pump, an empty gas tank, or a blocked fuel line. For the ignition system, these would be a bad distributor, bad spark plug wires, or bad spark plugs. These symptom-fault sets are used to derive initial hypotheses, directing the mechanic to look at only appropriate places in the causal model.

If, in fact, mechanics are using these two types of knowledge structures during troubleshooting, then we can predict several changes in these structures as a result of experience, and from those, we can predict the processing differences that would result from these changes. First, we predict that through experience, a mechanic's set of symptom-fault sets increases and that the sets he already knows become more accurate. As a result of these changes, the mechanic should have better ways to index into the causal model, leading to more efficient searches for the correct failure. Second, the causal model should become more filled out with experience, both through the addition of components and/or systems that were previously unknown and through the addition of relationships and dependencies between the known components. The causal model, like symptom-fault sets, should also become more accurate. As a result of having a better causal model, a mechanic should be better able to systematically reason about the way the car works, allowing him to find engine failures more systematically and in more cases.

We did, in fact, see clear differences between students at different levels of experience reflecting exactly these changes in their knowledge structures. First, we saw evidence that both the organization and number of symptom-fault sets increased with experience. The advanced student seemed to know more symptom-fault sets than the novice, as evidenced by the larger number of hypotheses he was able to generate for each problem. In addition, the advanced student seemed to organize his symptom-fault sets differently than the novice, evidenced by the more systematic procedure he used for generating and testing hypotheses. The advanced student's procedure was to zero in on one of the engine's subsystems and then to consider which component of that system was faulty, while the novice did not differentiate between systems and components of systems in diagnosis. While for the novice, all faults are equal and an hypothesis at the component level was as likely to be selected as the first to investigate as an hypothesis at the system level, the more advanced troubleshooter seemed to organize his symptom-fault sets into two categories, each used for different purposes. One set pointed to faulty *subsystems* within the car

(e.g., fuel system, electrical systems) and was used early in diagnosis to zero in on the faulty subsystem, while the second set pointed to faulty *components of these systems* (e.g., the fuel pump, the battery) and was used to diagnose the problem within that system. Such a change requires that the mechanic also reorganize his knowledge about the car's engine in a more hierarchical way that differentiates between systems and components of systems. Figure 1 shows a portion of the novice and advanced student's organizations of the causal model of the engine.

We also saw evidence that content of the causal model changed with experience. The causal model of the more advanced students contained not only more knowledge about individual components, but also more knowledge about the interconnected nature of the engine's systems. The behavior of the students during troubleshooting illustrates these findings. Consider, for example, the behavior of the advanced student in Problem 1D. His reasoning went as follows:[1]

> The first thing you want to do, which is the easiest thing to do, is look and see if we have any fuel, because you gotta have fuel, air, and heat... Don't have fuel..The first thing I want to do is check the fuse...they're OK... hook this jumper lead to the bypass to the fuel pump...the fuel pump is running... check and see our connection up here to the energizer...going from the ECM up to the injector is OK...try to energize this solenoid by hand...check to see if we got any gas...all the lines are alright...got gas to the throttle body... my diagnosis is the solenoid is bad because everything else checks out.

The hypotheses generated by this student are in an order that reflects the multilevel and highly integrated organization of both his causal model and his symptom-fault sets. He first determined which of three possible systems of the engine was affected and then investigated its components and others that could impinge on the behavior of the system under focus. In fact, his primary focus was on the electronic (or computer controlled) influences on the behavior of the fuel pump and fuel injectors. This reasoning showed an awareness (reflected in the student's causal model) of the interdependencies between subsystems. His reasoning shows that he knows that systems (such as the fuel and electronic systems) may intersect at several points and that an apparently or possibly failed component in one system may reflect an action, or lack of action in another system.

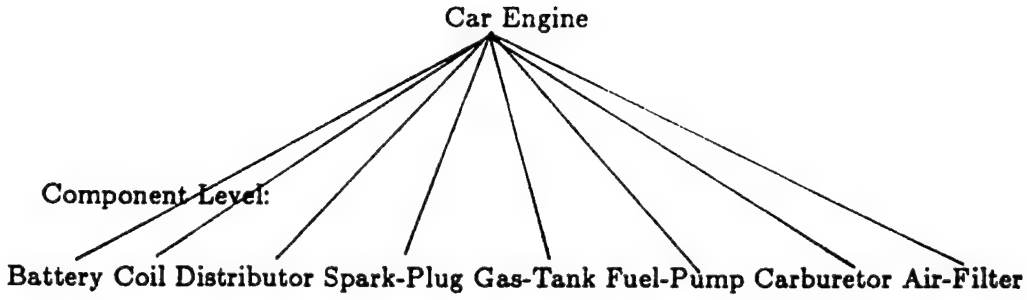
In contrast, the novice generated relatively few hypotheses for any given problem. His protocols indicate that this is because he has little knowledge about the relationships between given symptoms and their causes and also because his causal

[1] For a full protocol of the session, write to the second author.

NOVICE

Level of Abstraction *Values*

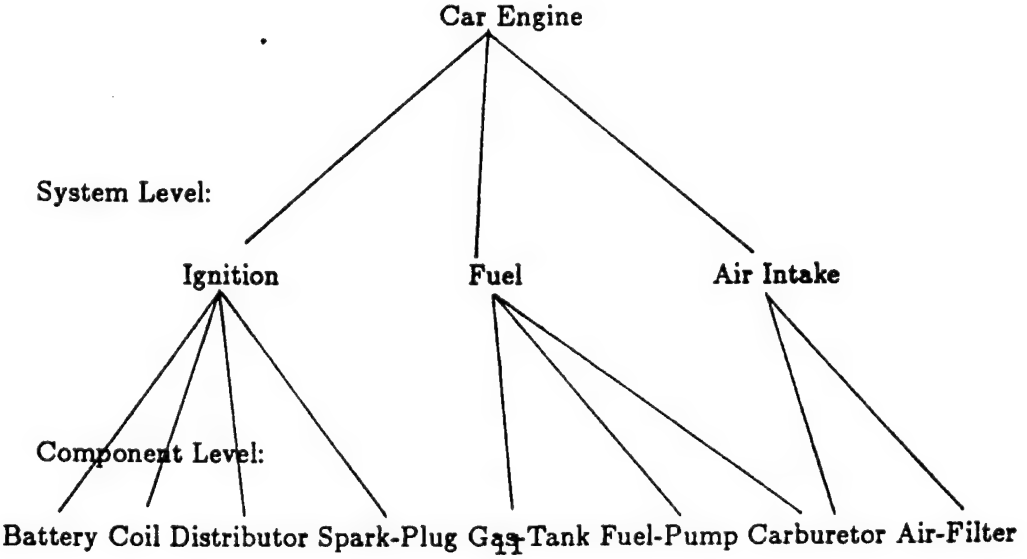
Highest:



ADVANCED

Level of Abstraction *Values*

Highest:



model is inadequate. In solving the same problem the advanced student was working on above, the novice reasoned:

> This problem could be in the fuel system, ignition system...we know it's not in the starting system because the car will crank over...One small drop of fuel...in that bowl...so it's in the fuel system...the fuel pump's... supposed to turn for 10 to 15 seconds...I can't hear it...It might just be a bad fuel pump.

We can see little evidence of an integrated hierarchy of levels in his organization of symptom-fault sets. While his hypotheses were sometimes at the system level (i.e., fuel system) and sometimes at the component level (i.e., fuel pump is bad), in only one problem (this one) did he clearly consider first a system and then a component within that system. More commonly, he generated hypotheses at both levels and then investigated only specific components. Furthermore, he showed a similar lack of integration in his causal model. Specifically, he never considered the possibility that one system could affect the behavior of another. His knowledge appeared to stop at the individual component's behavior and did not include the possibility that the actions of another system (the electronic system) could be affecting the behavior of the component he was considering (the fuel pump).

While the novice knew about many of the components of the car's engine and about what their connections were within a single system, he did not know how the systems and the components in different systems were interrelated. The advanced student, on the other hand, knew both the connections between components and the connections between systems. Thus the advanced student had a more integrated and complete understanding of the car's engine, while the novice's understanding seemed to be highly disjoint.

Figure 2 shows our interpretation of what the novice and advanced students knew about the fuel pump, for example. Note that the general information about pumps is available to both the novice and the advanced student in a declarative form. However, the information that the fuel pump requires an energy source which is the electrical system of the car is not part of the novice's representation of the fuel pump. If asked "What makes the fuel pump run?", the novice is able to construct the appropriate answer by using the more general information about pumps, but he does not use this knowledge during problem solving. The same pattern is probably true of knowledge about systems and components. The novice can undoubtedly tell an inquirer what system of the engine a particular component resides in, but he does not maintain this information where it is readily usable during problem solving.

PUMP Source: a container
Substance: a substance in the container
Conduit: a pipe
Destination: a container
Energy-Source: an energy device

NOVICE		ADVANCED	
FUEL PUMP	ISA PUMP Source: gas tank Substance: gasoline Conduit: hose Destination: carburetor	FUEL PUMP	ISA PUMP Source: gas tank Substance: gasoline Conduit: hose Destination: carburetor Energy-Source: electrical system

Figure 2: Novice and Advanced Student Representations of a Fuel Pump

We also saw within-subject changes in these knowledge structures over the course of the experiment. These changes were most evident in the intermediate student. Two examples will serve to demonstrate changes across problems. In working on problem three, the intermediate student made a long and protracted search for the fuel pump relay using both written reference materials and extended visual examination of the engine. While working on Problem 1D, he was able to immediately locate and check the same part. This component, and its physical relationship to others, had been incorporated into the causal model during or following problem three. Similarly, the symptom-fault sets changed as new information was acquired. For example, the first hypothesis the intermediate student checked at the component level for problem four was the fuel pump fuse, which was the correct diagnosis for problem three. He made the point as he worked that he was checking this possibility out first because of the previous case. ("I'm gonna check the fuel pump fuse *first* [this time].")

In addition to seeing changes in the organization and content of the causal model and symptom-fault sets of the students over the series of problems, we also saw them developing some additional knowledge structures: sets of diagnostic strategies and a malfunction model of the engine.

The *set of diagnostic strategies* stored the mechanic's knowledge about which tests could be used to evaluate which hypotheses and which values for the test results indicated properly functioning components and which values indicated failed components or systems. A more complete discussion of the implications of having

this knowledge can be found in the next subsection.

The *malfunction model* of the car was similar to the causal model already discussed. However, its content reflected the likely behaviors of individual components, systems, and the entire engine in the context of a variety of known *engine failures*. This model enabled the advanced student to test his hypotheses directly against the knowledge of how the system should behave if his hypothesis were correct rather than needing to always compare the engine's actual behavior against the working model of the engine.

4.2 Diagnostic Strategies

In addition to the changes experience makes in knowledge structures and organization, we also saw differences in diagnostic style. Diagnostic strategies seemed to be used differently by subjects at different levels of expertise and evaluation criteria changed significantly with experience. Some of these changes are due to the development of better strategies for testing and confirming hypotheses with experience while others appear to result from the differences in the knowledge available for diagnosis as a mechanic gets more experienced.

The change in how the mechanics tested and confirmed hypotheses was striking. As the example above showed, the novice student was willing to accept an hypothesis when preliminary evidence could be interpreted as congruent with that hypothesis and not pursuing the task any further (i.e. "can't hear the fuel pump"). In contrast, the advanced student sought, for each hypothesis, specifically confirming or disconfirming evidence that was part of a causal explanation. While he was willing to select an hypothesis to pursue on the basis of preliminary evidence, he would not accept or reject it without causally based information (e.g., "the fuel pump's not running, now we have to find out why").

The changes in diagnostic strategies that resulted from changes in the knowledge structures were more apparent in the efficiency of diagnosis. As the causal model gets filled out, it allows the mechanic to pursue a longer systematic search through the engine and to evaluate information in more detail and with more concern for the real effectsof the behavior observed. At the same time, as the number and complexity of symptom-fault sets increases, long searches become less necessary, because the mechanic is able to index into his model in more, and more effective, locations.

These two types of changes in the mechanic's diagnostic strategies work together to produce the results we saw. As the mechanic gains experience making correct and incorrect diagnoses, he gains a sense of what kind and how much information is

"enough" to validate his opinions. In addition, as his causal model and symptom-fault sets become more complete and accurate, he is more able to select appropriate hypotheses for investigation and to continue invgestigating a problem to the point that only one hypothesis remains as a possible diagnosis. Consequently, the conditions under which he will accept an hypothesis as a final diagnosis will become more accurate and the path by which he reaches his diagnosis will become more efficient.

This result is clearly evident in protocols of the novice and advanced students. When the novice's working hypothesis was a that a particular component was faulty, he either accepted it or rejected it as the cause of the symptom. He never investigated other effects on or inputs to that component. For example, in Problem 1B, the failure was a dead battery cell which caused the car to crank very slowly. The novice based his diagnosis on the following information:

> First of all, we'll have to check this battery...it could be the starter... it could be the alternator...it could be a voltage loss...could be a dead cell in the battery...we've only got 10 volts in the battery–each battery cell is 2 volts and there's 6 cells in the battery, so dead battery cell.

Here we see the novice generating both system and component level hypotheses, but because his knowledge is not hierarchically organized, not pursuing them in that order. Rather, he looks first at the battery charge. Because it is low, he accepts the hypothesis of a dead cell. His diagnostic strategy does not require that he consider any hypotheses relating to why the battery might be low, such as a malfunction in another system.

In contrast, the advanced student generally collected more information before giving a diagnosis. If possible, he confirmed his diagnosis by visually finding the condition that created the symptom (i.e., the disabled fuse panel connection in Problem 1C). When that was not possible, he justified his diagnosis within his causal model. For example, in Problem 1B, the failure could not be confirmed by visual evidence. Instead, the advanced student reaches his diagnosis with the following information:

> ...check the starter draw...it's pulling enough down to get the starter to go alright...We put the battery under load, you can see the amps rising and it's charging the battery...So the alternator's working OK...what I believe we have is the cell is dead in the battery...Try the test on the VAT...As you see on the indicator is also showing that it needs charging for the battery is bad...So what we have here is a battery with a couple of cells dead, and it's a sealed battery and you cannot check the specific gravity with a hydrometer to check and see which one's dead.

He reached and justified his diagnosis by eliminating all other possibilities from his symptom-fault sets and the causal model. In other words, he tested and verified normal functioning of both the starting system ("it's pulling enough down to get the starter to go alright") and the charging system ("So the alternator's working OK"). These are the only two systems, other than accessories such as headlights and radio, that affect the level of charge in the battery. Consequently, according to the student's causal model, if the battery's charge is low and the starting and charging systems are functioning correctly, the only remaining component in which the failure can be located is the battery itself. In some types of batteries, this conclusion can be tested directly, but in the car used in this problem, the battery is sealed. Therefore, the mechanic must stop with his explanation rather than attempt to verify the diagnosis any further. In comparison to the novice, he selected his hypotheses more efficiently, first eliminating competing systems from consideration. In addition, he based his acceptance of the diagnosis on a full causal explanation rather than on superficial evidence.

4.3 Summary: What was learned

As reported above, all of the students seemed to possess the same types of knowledge structures: a causal model of the engine, symptom-fault sets linking particular symptoms to the failures they might indicate, and a collection of troubleshooting guidelines and procedures. However, the students at various levels of training differed in both the amount of knowledge they had and the organization of that knowledge. The novice's knowledge was unorganized, sparse, and sometimes wrong. The intermediate students had a better organization on their knowledge (e.g., they grouped parts in the same system together) and some misconceptions had been corrected, but it was still incomplete. The advanced students had a much more complete knowledge base, and many of the misconceptions of the novice and intermediate students were corrected. In particular, advanced students knew more about what failures normally look like than did intermediate students, who knew more about how the car was supposed to work than about its malfunctions. The novice knew almost nothing about malfunctions.

The novice student, having learned the rudiments of how a car works in classroom instruction, was primarily focused on acquiring the necessary organization of the causal model to allow systematic tracking through the engine in search of a failure. For example, diagnosing a fuel system restriction in the earlier problem, the novice moved directly from the symptom (car cranks but will not start) to the hypothesis that it was not getting enough fuel and concluded that the fuel line was restricted. In a later case with the same symptom, the novice student instead showed that he had subsumed the fuel restriction hypothesis under a more global

hypothesis of fuel system failure. He first checked the end point of the fuel system for evidence of failure and then began checking components within the system. Thus he had modified his knowledge base in an important way between cases. He had collected the fuel system components into a system, allowing him to consider the system as a whole as his first hypothesized failure source. As a consequence, his trouble-shooting behavior became more systematic as he only considered specific components of a system if the system as a whole was shown to be the source of the failure. This type of learning can be termed *knowledge reorganization.*

The novice also improved his problem solving skills. In solving Problem 1A, he did not take action to confirm his diagnosis. Rather he was content to accept the hypothesis on the basis of its being a possible failure matched to the symptom. After attempting to solve the problem himself, he listened to the instructor's explanation of how to diagnose the a car's problem. The instructor, as part of his explanation, showed that it is necessary to confirm any hypotheses that are made. In the second case he solved, the novice had learned that confirming hypotheses is a necessary component of problem solving. He was thus also refining his problem solving skills.

In contrast, the intermediate students, having established the basic hierarchical organization of his causal model and symptom-fault sets, were primarily engaged in adding new information to his knowledge bases. Specifically, they showed evidence of adding to the faults associated with a given symptom in their symptom fault sets, learning new procedures for diagnosis, and identifying the locations and functions of new components. For example, on an early problem, an intermediate student made a protracted search for the fuel pump relay. On a later case, he was able to locate and test the relay at the appropriate time with ease. Thus, we see that he had learned not only its location but also how it was connected within the fuel system. Similarly, he employed techniques on a later problem that he had apparently not known when solving an earlier problem, leading to more efficient testing of hypotheses and an increased likelihood of getting correct, useful information from his testing.

Advanced students engaged in yet another type of learning that we have termed *refinement.* They already knew most of the information needed to successfully solve every problem presented. The changes in their knowlege between cases reflected primarily a reordering of faults in the symptom-faults sets to reflect new probablilties of occurrence. In other words, after solving a case in which the correct hypothesis was one he considered late in his troubleshooting procedures, advanced students returned to that hypothesis earlier in the sequence on the later problem. None showed notable changes in the overall organization or content of their knowledge between problems.

In addition, the advanced students seemed to have an additional model available to them that the intermediate and novice students were missing: a model of a

malfunctioning car. Both the novice and intermediate students seemed to diagnose faults by first zeroing in on a system that might be faulty and then checking the behavior of the car against the expected behavior of a *working* car. What differentiated the novice's behavior from that of the intermediate students was that the novice might focus on a particular part, while the intermediate students focused on a system. The more advanced students (and the instructor), however, seemed to know what to expect in a *malfunctioning* car. In other words, they not only had a *working model* of the car, but they also had a *malfunction model* of the car. We can see this in the behavior of the students when the testing equipment was faulty (Problem 1B). The advanced student was the only one of the students who worked on this problem who could recognize that the readings he was getting from the test equipment were faulty. The intermediate student who worked on this problem was unable to differentiate a bad reading on test equipment from a faulty car.

5 Learning Processes

Most researchers studying learning have investigated unsupervised learning or learning where the teacher gives an example but no explanations. However, our observations show that much of the learning that goes on early in problem solving depends on a teacher to give an explanation of the procedures for solving a problem and the knowledge needed to solve it. The causal model describing how a car functions properly, the one describing what malfunctions look like, associations between symptoms and faults, and the problem solving procedures the instructor finds useful motivate an instructor's explanations.

Early in learning, students don't know enough to be able to learn exclusively from their own problem solving experiences. For example, the novice who tracks down the error to the fuel pump when it is the fuel pump relay that is faulty (i.e., the fuel pump's imputs are causing it to malfunction, not the pump itself) can learn only if he is given extensive opportunity to attempt to fix the problem or if a teacher intervenes to show him what he was doing wrong. And, when the student is a rank novice, the experience of trying to fix it himself may be so overwhelming that, in the worst case, nothing is learned, while in the best case, learning takes a long time. It might take several attempts, for example, trying to put in a new fuel pump before the student will start thinking that something else is the matter, and at that point, there are so many things that could have gone wrong during his previous attempts (e.g., the new fuel pump may be faulty, he may have connected it up wrong in a whole range of different ways) that he may only be able to track down the problem if he is lucky in his initial guesses.

Nor did our intermediate student seem capable of learning without supervision. While he didn't need help with simple diagnostic procedures, he was unable to differentiate between instrument readings produced by faulty parts and instrument readings produced by faulty instruments. In other words, his missing knowledge about what faults look like required that an instructor intervene to instruct him on that subject.

Not only did students need to know how things work (i.e., have a nearly complete *working model*) to learn in a completely unsupervised setting, but they also need to know how things can go wrong and what things are in the normal realm of possibility.The advanced student, who was by no means an expert, was able to do quite a bit more learning by himself because he knew both of these things. He, however, also needed help from time to time when the knowledge he needed to solve a problem by himself was missing or faulty.

Several learning processes were identified by examining the protocols:

Learning by understanding explanations: After the students had diagnosed each problem, the instructor demonstrated the correct, or optimal, diagnostic path for the problem. In doing so, he enumerated both the reasons for considering each hypothesis and the diagnostic and test procedures he was using. The students attempted to modify their knowledge to match the instructor's wherever his procedures, hypotheses, or explanations differed from theirs. We have termed this process *learning by understanding explanations* (Redmond & Martin, 1988, Martin & Redmond, 1988). It is a combination of learning by observing an expert (Mitchell, et al., 1985) and learning by being told. In learning by understanding, the learner has an opportunity to observe another person (in this study, the instructor) solving the same case while providing an ongoing explanation of the knowledge and processes used. The learner notices those points at which the instructor's knowledge differs from his own and modifies his knowledge to bring it into agreement with the instructor's. There are a variety of things that can be learned by this method: new organizational structures, as when the novice collected the fuel system components into a system; new diagnostic strategies, as when the novice learned to test system endpoints or when the intermediate student learned how to use trouble trees; new causal connections in the causal model; new test procedures; and refinements of any of these things.

A more complete explanation of this process can be found in (Redmond & Martin, 1988) or (Martin & Redmond, 1988). In short, a student learning by this method attempts to explain each of the things the teacher is doing, and then uses the teacher's additional comments and explanations to bridge the gaps in his understanding. Thus, if the teacher proposes a particular hypothesis given a particular set of symptoms, the student (internally) attempts to explain why

that is a good hypothesis. The student may not know why a particular hypothesis is appropriate, but the teacher's explanation allows him to fill in the gaps in his attempted explanation. Without the teacher's explanations, the best the student can do is to draw associations.[2]

Active gap filling: While engaged in diagnosis, a student would sometimes realize that he did not know a specific piece of knowledge about the behavior of a component, or the connections between a component and the rest of the engine that was needed to solve the current problem. For example, his diagnostic procedures might have told him that he needed to check the input to some component, but he might not have known what the input source was. To find the missing knowledge, he would consult an outside reference source, such as a book or a more experienced individual. The information thus obtained was then incorporated into his knowledge base and was available during later diagnostic sessions. This is the process by which the intermediate student learned the location of and connections to the fuel pump relay. While he knew that he needed to find the fuel pump relay, he was unfamiliar with the particular car he was diagnosing. He knew that manuals provided this kind of information and went to the manual to find out. He went directly to the appropriate schematic in the manual and then spent considerable time using that schematic as a map of the engine and eventually finding the part he was looking for. In other words, he used an outside source (in this case, the manual) to find out exactly the specific piece of information he was missing.

We have termed this learning process *active gap filling* in recognition of the intentional nature of the learning. To engage in active gap filling, the individual must have enough knowledge about the system, enough problem solving knowledge to know what he ought to be doing, and an adequate organizational structure to allow him to recognize where the gaps in his knowledge might be, or at least to realize when an apparent dead end in his diagnosis might be the result of a lack of easily obtained information rather than something more extensive. Possibly because of this requirement for a fairly complete and well organized knowledge base, active gap filling was primarily used by the intermediate and advanced students as a learning tool. Our protocols show that knowledge learned through active gap filling is remembered and available for use in later problem solving.

Learning from interpreting feedback: When a diagnostic procedure was not yielding the results the student expected, or when he could not interpret the

[2]This process may remind people of Mitchell et al's (1985) learning by observing an expert, built into the LEAP system. Learning by understanding explanations assumes that the learner has incomplete knowledge and that the task of the learner is to *augment* its knowledge. LEAP's method, as well as other methods employing EBL and EBG (DeJong & Mooney, 1986, Mitchell, et al., 1986), assume that the learner has complete knowledge, and allow the learner to *better package* that knowledge.

results from a test, students had to ask for help. In essence, learning from interpreting feedback is a combination of active gap filling and learning by understanding explanations. As in learning by active gap filling the student is aware of a gap in his knowledge. He may not, however, know what that gap is. And, while active gap filling is usually a process of finding out about some feature of an object, learning by interpreting feedback focuses on procedures: in particular, "what went wrong" with a particular procedure. Learning by interpreting feedback can result in correcting a faulty causal model, but more often involves correcting and refining knowledge about how to do things. This process is similar to learning by understanding explanations, since the instructor may provide a causal explanation of some phenomenon or offer advice about carrying out procedures. It is a more active and goal-directed process, however, initiated by some complication the student is experiencing while solving a problem.

Learning by interpreting feedback is invoked when the student cannot interpret the unexpected results of his problem solving. The process of explaining those results can be done by the student or by asking the instructor. When an intermediate student found that he was getting test results that he could not interpret he went to the instructor for help. On the same problem, the advanced student figured out for himself that the test equipment was faulty, and was able to learn on his own which ranges of test results predicted faulty equipment. The advanced student could do this, while the intermediate one could not, because he had knowledge telling him what things look like when they malfunction. This experience told him what malfunction of a particular instrument looks like. The intermediate student, on the other hand, did not know enough to hypothesize by himself that the test equipment might be faulty. On another occasion, however, the advanced student needed the instructor to provide an explanation to him. He was trying to energize the fuel pump from the battery using a test light to connect them together. He could not get the fuel pump to go on. He, like the intermediate student in the previous example, knew what the results should look like (in this case, he thought the fuel pump should go on) but did not know why the results he was getting were different. He asked the instructor what he was doing wrong. The instructor told him that he needed to use a lead (a wire with no bulb attached) to energize the fuel pump, since with a light attached, the bulb consumes the power from the energy before it gets to the pump. In this case, the advanced student's causal model of electricity was faulty.

Abstraction: Another learning process that was noted was *abstraction.* It seemed that any new information acquired by a student between problems was incorporated into the knowledge base at several different levels. We can see this most clearly in the novice student's behavior. Recall that in one problem, he tracked the failure to the fuel pump but did not examine the system further to distinguish

whether the problem was in the fuel pump or in the input to the fuel pump. The instructor followed that session with an explanation of how to diagnose the fuel pump problem. In his explanation, he stated that the endpoints of the fuel system had to be checked for evidence of a problem. The student learned this (by understanding the instructor's explanation) and in the next problem applied it. The student also apparently learned the abstract principle the instructor was illustrating: that system endpoints need to be checked for problems when diagnosing any system. Abstraction is sometimes done spontaneously, as in the example just given, and is sometimes induced by the instructor during explanation. For example, in one instance the instructor walked students through an engine they hadn't seen before, and stated that parts with a particular function had to be found. He then explained what those parts look like in general.This abstract knowledge allows the student to identify the part no matter what type of car he is looking at.

Case-Based Reasoning: A final process that facilitated learned was *case-based reasoning.* In case-based reasoning, a previous case that a problem solver is reminded of provides an answer to a new problem or focuses him on the knowledge needed to solve the problem (Kolodner & Simpson, 1984, Hammond, 1986). While in the previous paragraphs, we have referred to learning processes that allow a problem solver to learn new facts, case-based reasoning is a problem solving method that allows a problem solver to improve its performance without full understanding of the facts. Remembering previous cases allows a problem solver to solve a new problem better than an old one even when the problem solver is missing a causal explanation of why the previous solution did or did not work. Our hypotheses about case-based reasoning in experts predict that those cases that are different than what is expected are the ones that are remembered (Kolodner, 1982, 1983, Kolodner & Simpson, 1984, Schank, 1982). But novices don't always know the norms, so they can't recognize that something is different.

The data we collected in this study show three situations in which novices stored cases for later problem solving. First, if a case serves as a strong justification for doing a procedure, then the case was "kept" and referred to later. The advanced student, in an early problem, spent 45 minutes trying to solve a problem that he could have solved instantly with visual inspection. In the next problem, he did a visual inspection immediately, saying "what we want to do is check the wire...like we did the last time".

Second, cases were maintained in memory if they served as an example of a particularly complex or unusual situation. This includes, among other things, cases where a set of symptoms predict a highly unusual fault and cases where some set of symptoms predict a fairly commonplace fault that is hard to diagnose. Both students who solved problem 2C (one intermediate and one advanced), in which a rocker arm had been removed, for example, were able to do it on the basis of

remembering a previous case in which a rocker arm had been broken. A broken rocker arm is a highly unusual problem. In almost all cases, rocker arms outlive the cars they are in. The other students, who had never seen such a problem, were unable to solve the problem (one intermediate and one novice). These two students were able to determine that none of the possible common causes of the symptoms were responsible for the failure of the car, but they were not able to pursue their investigations beyond that point. The two successful students were reminded (based on the particular sound of the car) early in their diagnosis of a car they had worked with in the previous quarter that had the same problem. This reminding led both of them to try the diagnostic procedures that would lead them to the correct diagnosis and, thus, to the diagnosis.

A third instance in which a previous case was kept was when it illustrated something that was not known previous to that case. The case was remembered until what was learned from it was confirmed by a later case. While one of the other procedures might have been used to learn a general concept, the case in which it was learned seemed to remain available to the students until the new item of information was confirmed by another case. We could not determine, however, whether the general knowledge or the case was accessed first in later problem solving.

6 Summary and Conclusions

We have attempted to outline the knowledge that student problem solvers at various levels of expertise have available to them as they solve real world problems and the learning procedures by which they augment and refine that knowledge.

Diagnostic behavior seems to require the interaction between two major types of knowledge structures: *causal models* and *symptom-fault sets*. The knowledge and organization of these knowledge structures changes with experience. In the causal model, the most notable change is the increasing complexity of the model, reflected in the growing awareness of the interconnectedness of systems within the engine. The novice is clearly unaware of the possibility that electronic failures can affect things like fuel delivery, since he knows little about the dependencies between the fuel system and the electrical system, while the more advanced mechanic not only knows that such relationships exist, he considers them a highly common source of failures. Similarly, the number, organization, and accuracy of the symptom-fault sets changes with increasing experience. Ultimately, they are able to represent a complex, hierarchical system of relationships. The data suggest that components are organized hierarchically under their respective systems and are never directly considered unless their system is determined to house the failure, or at least to be the source of information crucial to locating the failure.

Building partly on these changes in the knowledge structures, and partly on independent effects of experience on decision processes, the mechanic's procedures and guidelines for accepting hypotheses as diagnoses also change. The processes or procedures used become increasingly focussed on information that will allow a causal interpretation of the behavior observed. At the same time, the developing knowledge structures allow the mechanic to search for and aquire more, and more accurate, information from his symptom-fault sets and his causal model. The interaction of these changes in both knowledge and process lead to the more accurate and efficient problem solving seen in experts.

Thus, we see that experience is providing the mechanic with three things. His overall level of knowledge is increasing; the organization and integration of his knowledge structures, both the symptom-fault sets and the causal model, are increasing; and his processes and criteria for reaching diagnoses are becoming more accurate, more efficient, and more focussed on causal information.

We have also identified five different real-world learning procedures used by the students and several roles a teacher must play in helping a student to learn at various stages. *Learning by understanding explanations* is the process by which students integrate a teacher's explanation of how to solve a problem with their own diagnostic knowledge. *Active gap filling* allows a student to fill in known gaps in his knowledge by asking questions or looking in source books. *Learning from interpreting feedback* is used when the student is unable to evaluate test results he has obtained. Interpretation may or may not require intervention of a teacher, depending upon the student's knowledge state. *Abstraction* lets the student reorganize his knowledge in better ways. It can be done independently by a student or pointed out by a teacher. *Case-based reasoning* allows the student to improve his problem solving without having a full understanding of how things work. The cases students remember might be their own attempts to solve problems or the explanations given by a teacher of how to solve a problem. We have identified three circumstances under which students seem to retain their experiences: when an experience was unexplainable, when it provided the first introduction to some concept,and when a serious mistake was made.

While much research has gone into unsupervised learning, little research has focused on processes by which a student learns from a teacher. Among the learning processes we have identified,three require extensive interaction with a teacher: learning by understanding explanations, active gap filling, and learning by interpreting feedback. The particular interactions depend on the knowledge the student already has. More research in this area is surely needed if we want to develop better teaching technologies and practices.

7 Bibliography

1. Bhaskar, R. & Simon, H.A. (1977) Problem solving in semantically rich domains: An example from engineering thermodynamics. *Cognitive Science*, 1, 193-215.

2. Chi, M.T.H., Glaser, R., & Rees, E. (1982) Expertise in Problem Solving. In R.J. Sternberg (ed) *Advances in the Psychology of Human Intelligence.* Hillsdale: Lawrence Erlbaum.

3. DeJong, G. & Mooney, R. (1986). Explanation-based learning: An alternative view. *Machine Learning*, Vol. 1.

4. Glaser, R. (1985) *Thoughts on expertise.* Technical Report #8. Learning Research and Development Center, University of Pittsburgh, Pittsburgh, PA 15260.

5. Hammond, K. (1986). *Case-Based Reasoning: An integrated theory of planning,learning and memory.* Ph.D. Thesis. Yale University.

6. Hunt, R.M. (1981) *Human pattern recognition and information seeking in simualated thought diagnosis tasks.* Report #T-110, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign.

7. Kolodner, J. L. (1982). The Role of Experience in Development of Expertise. *Proceedings of AAAI-82.*

8. Kolodner, J. L. (1983). Maintaining Memory Organization in a Dynamic Memory. *Cognitive Science*, vol. 7.

9. Kolodner, J. L. & Simpson, R. L. (1984) Experience and Problem Solving: A Framework. *Proceedings of the Sixth Annual Conference of the Cognitive Science Society.*

10. Lancaster, J. S. & Kolodner, J. L. (1987). Problem solving in a natural task as a function of experience. *Proceedings of the Ninth Annual Conference of the Cognitive Science Society.*

11. Lancaster, J. S. & Kolodner, J. L. (1988). Varieties of Learning from Problem Solving Experience. *Proceedings of the Tenth Annual Conference of the Cognitive Science Society.*

12. Martin, J. D. & Redmond, M. (1988). The Use of Explanations for Completing and Correcting Causal Models. *Proceedings of the Tenth Annual Conference of the Cognitive Science Society.*

13. Mitchell, T. M., Kellar, R. M. & Kedar-Cabelli, S. T. (1986). Explanation based learning: An unifying view. *Machine Learning*, Vol. 1.

14. Mitchell, T. M., Mahadevan, S., & Steinberg, L. I. (1985). LEAP: A Learning Apprentice for VLSI Design. *Proceedings of IJCAI-85.*

15. Rasmussen, J. (1978) *Notes on diagnostic strategies in process plant environment.* Riso National Laboratory Report #RISO-M-1983, Roskilde, Denmark.

16. Rasmussen, J. (1979) *On the structure of knowledge: A morphology of mental models in a man-machine context.* Riso National Laboratory Report #RISO-M-2192, Roskilde, Denmark.

17. Rasmussen, J. and Jensen, A. (1974) Mental procedures in real life tasks: A case study of electronic troubleshooting. *Ergonomics*, 17(3), 293-307.

18. Redmond, M. & Martin, J. (1988). Learning by Understanding Explanations. *Proceedings of the 26th Annual Conference of the Southeast Region ACM*, Mobile, Alabama.

19. Schank, R. C. (1982). *Dynamic Memory.* Cambridge University Press.

Appendix B

GIT-ICS-89/17

MECH:
A Computer Interface for Teaching and Investigating Mental Models and Troubleshooting

Lawrence W. Barsalou[1]
Christopher R. Hale[1]
Michael T. Cox[2]

[1]School of Psychology
Georgia Institute of Technology
Atlanta, GA

[2]microSpheres
Atlanta, GA

March 1989

This project was supported by Army Research Institute contract MD A903-86-C-0172 and was written while the first author was a visitor at the University of Michigan. We are grateful to Janet Kolodner for the opportunity to perform this work. Correspondence should be addressed to Lawrence W. Barsalou, School of Psychology, Georgia Institute of Technology, Atlanta, GA 30332 or barsalou@gatech.edu.

Abstract

MECH is a computer interface for providing instruction on mental models and troubleshooting skills and for performing research on these topics. Section I describes MECH's general characteristics. MECH currently runs on IBM PC class computers, is relatively domain-independent, and does not require programming to use. Instead, authors must only construct text files with a text editor to configure MECH for a particular domain. Using hypertext and hypergraph mechanisms, MECH provides students with structural and functional knowledge of a domain at a conceptual level. MECH does not currently provide high-fidelity experience, nor simulate the system it models. Although MECH does not simulate a domain, it uses simple, easily-constructed production systems to provide a dynamic and realistic troubleshooting environment. MECH also teaches functional/qualitative reasoning, symptom-fault rules, and domain-independent strategies that underlie troubleshooting. MECH does not currently teach actual test and repair procedures. Section II presents the theoretical principles that underlie MECH's design. Section III presents an overview of MECH's utilities and how they work. Sections IV and V describe how MECH can be configured for a wide variety of instructional and experimental settings. MECH stores a complete record of keystrokes, millisecond latencies between keystrokes, and screen information from each session. These keystroke files provide a rich source of data and can also be used to control MECH during later sessions in a wide variety of manners. MECH has a payoff structure that can be used to motivate students and measure their troubleshooting ability. MECH also includes a timed true-false test that can be run optionally. Section VI describes how to obtain the MECH software at no cost and configure it. This section also provides complete instructions on how to perform demonstration sessions and how to perform all possible operations within MECH. Section VII provides complete instructions on how to author files for a new domain and on how to alter already existing files. Section VIII describes numerous suggestions for further developing and improving MECH through reprogramming.

I. Introduction

A. What MECH Is

MECH--a computer program currently developed for IBM PC class computers--provides students with training on the structure and function of conceptual domains. Consider the domain of small lawn mower engines, for which we have currently configured MECH. Through graphics and text utilities that decompose domain structure, MECH teaches the hierarchical organization of the engine. For example, MECH teaches the systems that compose the highest level of engine structure, namely, the fuel, ignition, cylinder, drive train, lubrication, and cooling systems. In turn, MECH illustrates the composition of these systems. For example, the ignition system decomposes into the magneto, breaker points, and spark plug. Decomposition continues until MECH reaches the level of terminal components. For example, the breaker points decompose into terminal components for the plunger, moving arm, spring, moving point, and stationary point.

MECH also teaches students functional organization, showing the functional/qualitative relations underlying each system. For example, MECH might show how air and fuel enter the carburetor, how they are mixed in the carburetor, and where the carburetor sends air-fuel mixture.

Note that MECH only provides students with an abstract conceptual account of a domain and does not provide direct sensori-motor experience. Students who learn about small engines from MECH would probably not know what a carburetor looks like or how to find one in an actual engine. Instead students using MECH would acquire *conceptual understandings* of the carburetor's internal structure and function, as well as how it interacts with other parts of the engine. In Section VIII.K, we suggest how MECH could be extended to provide sensori-motor training, along with conceptual training.

There are a wide variety of applications across which such a conceptually-oriented tutor may be useful. Some possibilities are as follows:

(1) In instructional domains, students may often need to acquire a conceptual understanding of how something works. For example, a student taking a non-performance course on mechanical devices may need to acquire abstract accounts of device structure and function.

(2) A manager may want a rudimentary understanding of how the equipment works that his or her assistants operate.

(3) Someone who does troubleshooting completely according to fixed procedures ("by the book") may want a conceptual understanding of the domain simply out of curiosity.

(4) Someone who does troubleshooting in a domain where fixed procedures do not exist, and where functional/qualitative reasoning is necessary for isolating faults, may find conceptual tutoring, not only helpful, but necessary in preparing for troubleshooting.

session can be used to show a second group of subjects the second technique. Both groups can then perform troubleshooting on a common problem set to see which technique promotes the best learning.

To run MECH under "keystroke" control, a student simply presses the <ENTER> key continually. Each time the student presses <ENTER>, MECH implements the next keystroke from the keystroke control file, just as if it had been entered at the keyboard. A wide variety of manipulations can be constructed through this mechanism.

(4) MECH contains a timed true-false test that can be performed at the end of a session, if so desired. Responses are measured with millisecond accuracy.

(5) MECH presents students with extensive feedback about the success of their troubleshooting efforts, as well as associated costs. This payoff structure can be used to motivate subjects in various ways and to measure individual differences in troubleshooting skill.

Turning to implementation, MECH has been designed for use across a wide range of domains. We have identified what we believe are domain-independent principles of mental models and troubleshooting and have developed MECH around them. Although we have currently configured MECH for the domain of small engines, it could be configured for a wide variety of other domains, including other mechanical devices, electronic devices, social organizations, computer programs, and so forth. In fact, our programmer has configured MECH to teach programmers about itself, so that programmers can easily make alterations if they wish. If a reader is interested in obtaining this particular implementation of MECH, they can do so by writing us and including a floppy disk (see Section VI.A for disk requirements).

However, MECH does *not* require any programming to be used across a wide range of domains and applications. If a user wants to configure MECH for some other domain besides small engines, he or she only needs a text editor to construct the input files that MECH requires. Section VII completely describes how to construct every input file that MECH uses. Similarly, if a user wants to revise our input files for small engines, Section VII provides all the necessary information about how to do this. If a user is interested in reprogramming MECH, Section VIII contains some relevant guidelines.

B. What MECH Is Not

MECH does not provide students with a high-fidelity simulation of a system. Students do not receive perceptual training that would allow them to actually identify the components of a system or malfunctioning components. Similarly, students do not receive procedural training in actually performing tests and making repairs. For these reasons, MECH is not sufficient for teaching troubleshooting, although it may be a useful accompaniment to hands-on training or to a high-fidelity simulator. Because hands-on training and high-fidelity simulators provide so much sensori-motor information, students may have trouble simultaneously extracting and

II. MECH's Design Principles

In this section we present the theoretical assumptions that underlie MECH's design. Please note that we do not cite much previous work relevant to our interests. Instead our goal here is primarily to describe the system we have constructed. Later reports will make more contact with the literature.

A. Representing Mental Models

We assume that the following representational assumptions apply to most mental models:

1. Analogical mapping. People generally try to acquire a one-to-one mapping between the components and relations of their mental model and the components and relations in the corresponding physical device (Johnson-Laird, 1983). A person's mental model may not represent all the components and relations in the corresponding physical device. Components and relations that are represented may be incorrect.

2. Hierarchical organization. To the extent that the organization of a physical device is hierarchical, the organization of a mental model may typically be hierarchical as well. More specifically, the device and the mental model both decompose into high-level systems, which in turn decompose into lower-level systems, and so forth, before decomposing into terminal components. Components belonging to two systems may occasionally violate strict hierarchical organization. For example, the spark plug can be viewed as belonging both to the ignition system and the cylinder assembly. Even with these violations, it is still possible to decompose a device in a quasi-hierarchical manner that serves as a useful organization of components. In a small engine, decomposition may proceed from the engine to the ignition and fuel systems, from the ignition system to the magneto and spark plug, from the magneto to the coil, and from the coil to terminal components such as the primary and secondary wires.

3. Multiple models. A given mental model may be one of many possible for the same physical device. There are many ways models can differ. There can be simple ways in which different people represent the same device somewhat differently. Or there may be multiple models for the same device that capture fundamentally different kinds of input/output relations and serve fundamentally different kinds of reasoning goals (e.g., White and Fredericksen, 1986).

We assume that any hierarchically-organized mental model contains the following three types of representational entities:

4. Components. These include representations of the specific systems that compose a more general system, as well as the terminal components of the most specific systems. In a small engine, components of the engine include the fuel system and ignition system (decomposition of a general system to more specific systems); components of the coil include the primary wire and secondary wire (decomposition of a system to terminal components).

changes in one component affect the qualitative states of connected components (e.g., as the the throttle valve is increasingly opened, increasingly large amounts of fuel enter the carburetor).

As numerous theorists have suggested, people use functional/qualitative rules to simulate performance of a device. Series of rules can be applied to see how the input to one component produces effects over paths of relations that emanate away from it. Following Hegerty, Just, and Morrison (1988), the operation of these rules depends on the properties of the respective components, as well as on the inputs they receive (e.g., the rate of fuel flow through a tube depends on its diameter, as well as on the amount of fuel it receives as input). Both properties and inputs provide constraints on the behavior of components. Functional/qualitative reasoning rules capture these constraints and enable predictions about performance. More specific forms of these rules may on occasion allow quantitative prediction.

2. Symptom-fault rules. These rules start with observed problematic symptoms and provide hypotheses about what components might be at fault. For a small engine, a symptom-fault rule might state that whenever there's a strong gas smell during engine operation, check to see if the choke is broken. Another rule might state that whenever the engine type is Briggs and Stratton, check to see if the condenser is broken. Symptoms may either be causally related to faults, or they may be incidental features that correlate with faults.

3. Meta-rules. These include rules about how the structure of a mental model should be searched to find a fault (e.g., breadth-first versus depth-first); rules about the transitivity of qualitative reasoning (e.g., if component *X* produces an input to component *Y*, and if component *Y* produces an input to component *Z*, then *X* produces a distant input to *Z*); rules about how to handle remindings; etc. In contrast to qualitative reasoning and symptom-fault rules, meta-rules may be fairly domain-independent. In general, meta-rules guide the executive control of troubleshooting by setting goals, deciding how to handle errors, handling interruptions and unexpected results, etc. (cf. Norman & Shallice, 1986).

An organizational principle also seems important to processing:

4. Compilation. With practice at using any sequence of the above rules repeatedly, the sequence may become compiled into a procedure that produces more efficient processing in the future. Sequences of qualitative reasoning rules may become automated for frequent kinds of qualitative reasoning. Sequences of symptom-fault rules may become automated to zero in quickly on suspected faults. Sequences of meta-rules may become automated to minimize wasted resources and non-optimal behavior. Moreover, combinations of different types of rules may become automated to the extent they frequently occur in a systematic pattern.

The above seven principles of representation and four principles of processing guided our design of MECH. As described in later sections, MECH can be used, at least to some extent, for training students on each of these principles. Again, by no means do we claim that MECH is sufficient for providing students with complete understanding or skill in any domain.

Figure 1
MECH's Menu System

```
Current System: engine\ignition\magneto                    Menu: Main

   F1  Move                                   F6  Help
   F2  Tutor
   F3  Test
   F4  Repair
   F5  Jobs
```

```
Current System: engine\ignition\magneto                    Menu: Move

   F1  Move Up to Engine System
   F2  Move Up One System
   F3  Move Down to Subsystem #
   F4  Follow an Input
   F5  Follow an Output                       F10 Return to Main Menu
```

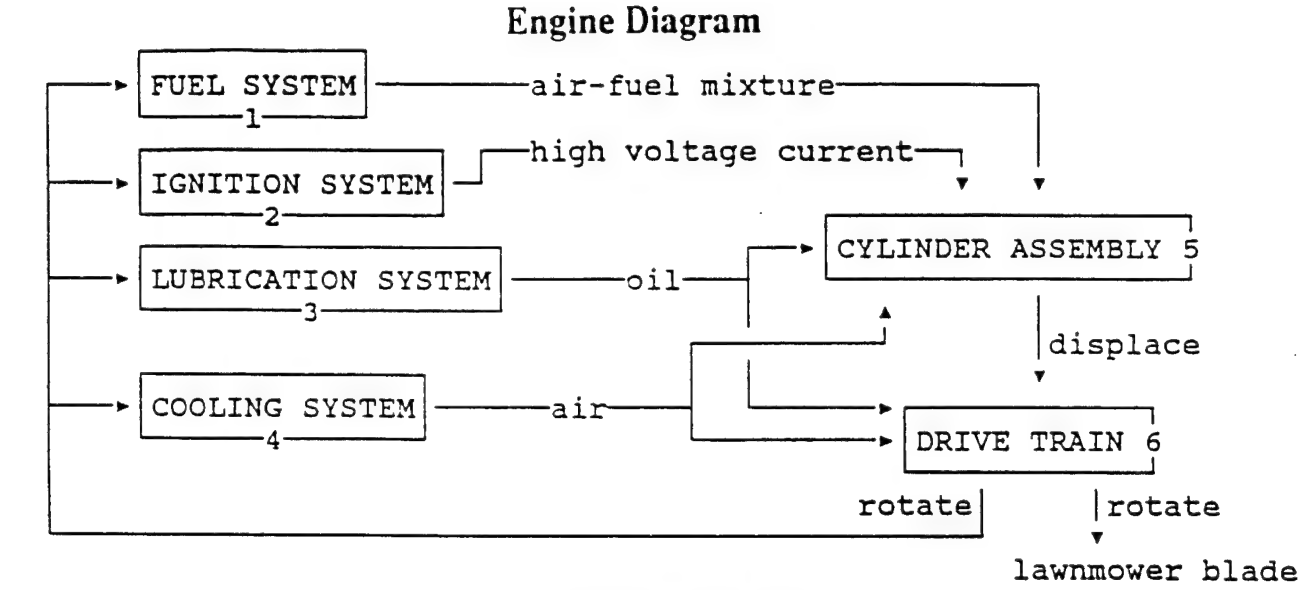
```
Current System: engine\ignition\magneto                    Menu: Tutor

   F1  System Description
   F2  Description of Component #
                                              F10 Return to Main Menu
```

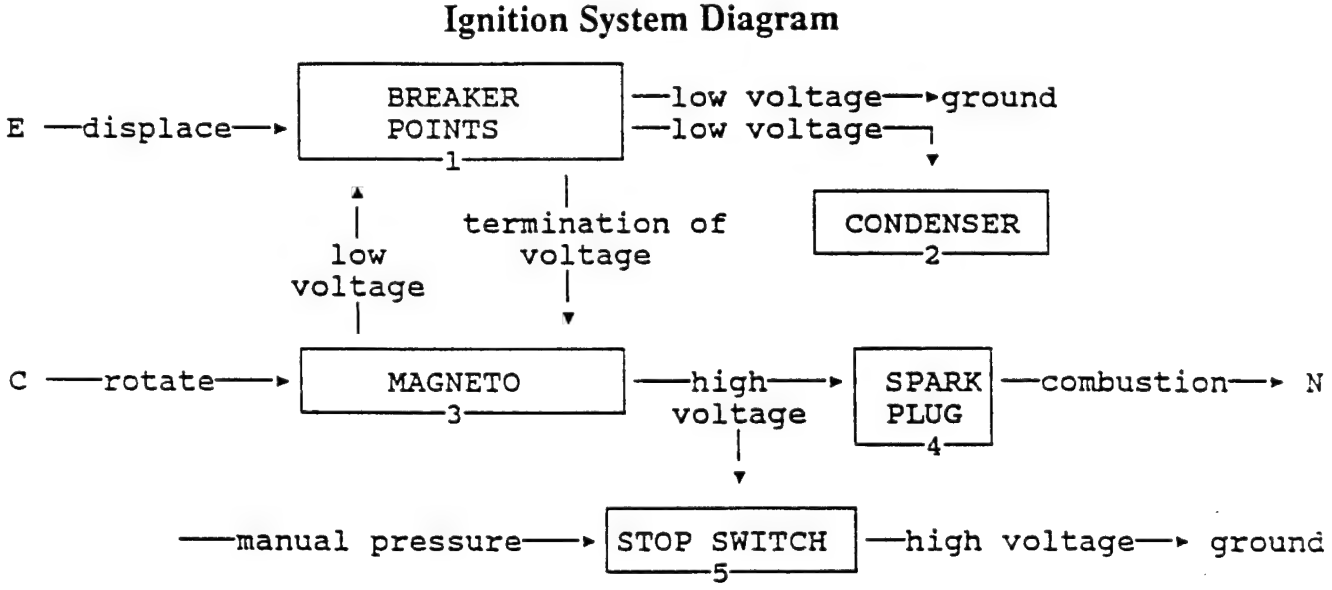
```
Current System: engine\ignition\magneto                    Menu: Jobs

   F1  First/Next/Current Job
   F2  Previous Job
                                              F10 Return to Main Menu
```

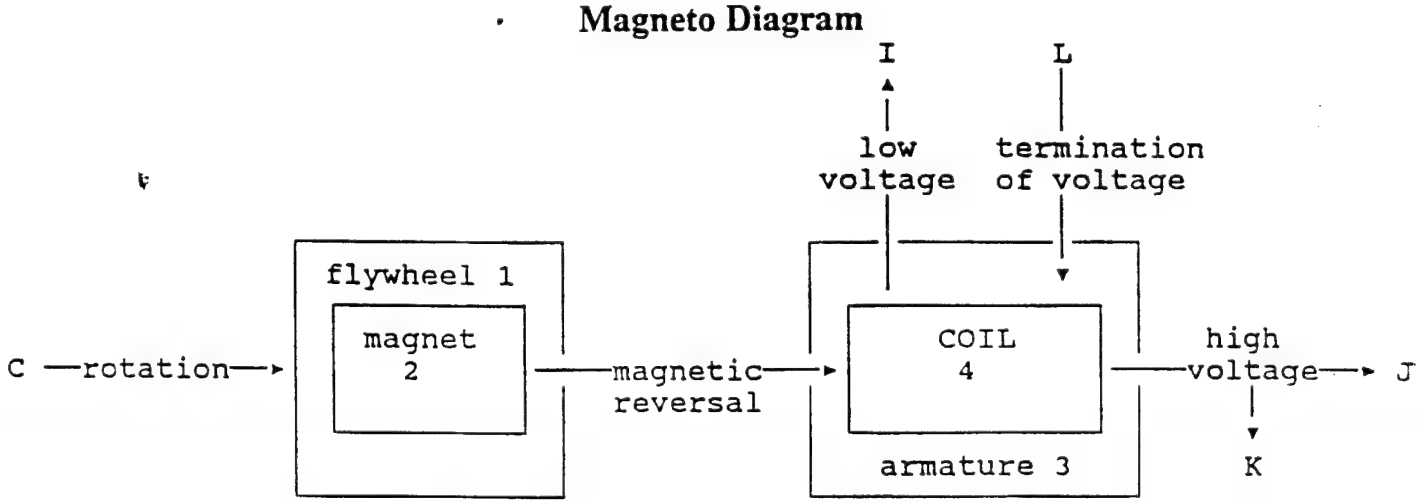
```
Current System: engine\ignition\magneto                    Menu: Test

   F1  List-Perform System Tests              F3  Start the Engine
   F2  List-Perform Component Tests           F4  Stop the Engine

                                              F6  List Charges So Far
                                              F10 Return to Main Menu
```

```
Current System: engine\ignition\magneto                    Menu: Repair

   F1  List-Perform System Repair             F3  Start the Engine
   F2  List-Perform Component Repair          F4  Stop the Engine
                                              F5  List Previous Repairs
                                              F6  List Charges So Far
                                              F10 Return to Main Menu
```

Figure 2
Examples of Nested Diagrams for a Small Lawnmower Engine







Currently, MECH does not use bit-mapped diagrams. Instead its diagrams must be constructed in ASCII and extended ASCII (see Section VII.A.4). As discussed in Section VIII.D, future versions of MECH will incorporate bit-mapped graphics as an option.

The MOVE module allows students to move quickly and easily through the space of diagrams. Except in a few situations, a diagram is always present on the screen. To move to a new diagram, students call MOVE and implement one of the functions from its menu. These functions include moving to the root diagram for the engine, moving up one level in the hierarchy from the current diagram, or moving down to a lower system. Students can quickly and clearly view decomposition of the engine using these three "vertical" functions. Two further functions allow traversing external input and output relations. By tracing back through the inputs to a system, students can discover how various forces and substances converge on that system. Following the outputs out of a system allows students to see how the current system affects other systems. Students can quickly and clearly view the functionality of the engine using these two "horizontal" functions.

Using MOVE is essential to the TUTOR, TEST, and REPAIR utilities. Tutoring, testing, and repairing at any point during program use are always limited to the current diagram. For example, if a student is currently viewing the diagram for the breaker points, he or she can only learn about, test, or repair the breaker points. To learn about, test, or repair some other part of the engine, the student must move there first.

An advantage of this design is that it allows us to track what parts of the system a student is currently examining. Because MECH records all keystrokes (described in Section III.E), we can follow a student's path through the model. We can see when they needed tutoring, when they ran tests, and when they attempted repairs. Because we also store the latency for each keystroke, we can see how much time a student spent at each point in the model performing an operation. Sections III.E and VI.J discuss keystroke data files in greater detail.

3. TUTOR. The diagrams provide some tutoring, given they represent the components and relations for each system. TUTOR, through text, can describe these components and relations in much greater detail. Analogous to the diagrams, text units are organized hierarchically. The root contains the text describing the engine as a whole at a very general level. Texts associated with subordinate nodes of the engine topology describe increasingly specific systems that constitute decomposition of the engine.

For each diagram, students can access one *system description* of the diagram as a whole and one *component description* for every component in the diagram. The system description provides a general description of how the system functions as a whole and how it interacts with external systems. Each component description describes the structure and local function of a specific component in the current diagram. Section VII.B provides detailed information about the descriptions used by TUTOR.

Whenever a student performs a test, he or she receives feedback about the results. The test may have been unnecessary, or it may have any number of other possible outcomes. The outcome of a test depends on *productions* stored in a file written specifically for the current job. This file, and the productions in it, are easily constructed with a text editor. Section VII.D.2 describes these productions in complete detail. More generally, each production has a set of *broken conditions* and a set of *working conditions* that trigger it. Broken conditions are components that must be broken for the production to fire (i.e., faults that haven't been repaired yet). Working conditions are components that must be working for the production to fire (i.e., faults that have been repaired). If both sets of conditions are completely met, the message associated with the production is presented to the student as the outcome of the test.

One advantage of this approach is that the outcome of a given test can vary widely, depending on the current state of other systems and components in the engine. For example, imagine that there are obstructions in the carburetor, both for air and fuel intake. Further imagine that a student tests the cylinder to see whether it is admitting air-fuel mixture. If neither the air or fuel intake has been repaired, students receive a message saying that nothing is entering the cylinder; if the air problem has been repaired, students receive a message saying that only air is entering; if the fuel problem has been repaired, students receive a message saying that only fuel is entering; if both have been repaired, students receive a message saying that the test revealed no fault.

As can be seen from this example, a test can report that a component is not working properly even though it is not a fault (e.g., the intake value of the cylinder may report not receiving air-fuel mixture, not because the valve is broken, but because other components connected by external relations are broken). By taking advantage of such relations, we can orient students toward causal reasoning about symptoms and faults. Moreover, because a given problem takes both broken and working conditions into account, it provides a dynamic testing environment in which the outcomes of tests can vary over the course of troubleshooting. Sections VII.C and VII.D provide a complete account of how MECH handles tests and repairs during troubleshooting.

6. REPAIR. A student is free to make any repair at any time, regardless of whether it needs to be done. If the repair was necessary, students receive a message saying that the repair was needed. If the repair was unnecessary, students receive feedback saying the repair was unnecessary. When a repair is made, the fault disappears, thereby changing which test productions will fire. To see if the current engine has been completely repaired, the student can try to start the engine at any time. If no faults remain, then no productions will fire. MECH will state that the engine has been repaired and that the student should go on to the next problem. Students must try to start the engine after every repair before attempting any additional tests or repairs.

D. Payoffs During Troubleshooting

A cost is associated with every test and repair. Section VII.C describes how the author of a MECH configuration assigns costs. Students receive constant information about how much

E. Keystroke Data Files and SHOWKEYS

Every time MECH runs, it stores a complete record of all keystrokes and their millisecond latencies. For each keystroke, MECH stores the current diagram on the screen, the current menu on the screen, the particular options selected from the menu, and the time since the previous selection from a menu. MECH stores this information originally in a compact file that is somewhat difficult to read (by humans). Consequently, we wrote SHOWKEYS, which transforms a keystroke data file into a larger file that is easily readable. Once one has become familiar with the diagrams and menus in a domain, one can read a file produced by SHOWKEYS easily, thereby following what the subject did.

F. Keystroke Control and Split Control

MECH is designed such that the keystrokes provided by one user can be used to control MECH for other users. At the start of each session, MECH asks if the session will use keystroke control. If keystroke control is used, MECH loads a keystroke file constructed from a previous session. To operate MECH, a student simply presses the <ENTER> key to present each new screen of information. Upon receiving each <ENTER>, MECH implements the next keystroke in the keystroke control file, just as if it had been entered at the keyboard.

Keystroke control can be used in a number of ways. For example, an instructor could present high-level systems to students on one occasion and low-level systems on another. Similarly, an instructor could demonstrate breadth-first search for a fault on one occasion and qualitative reasoning on another. Using keystroke control, an instructor can present any part of MECH's data files to a student, combined with any of MECH's operations, in any order. Experimentally, keystroke control can be used to implement a large variety of independent variables. Sections IV and V describe various possibilities concerning keystroke for instruction and research.

MECH also allows *split control*. When split control is invoked at the start of a session, the student starts out under keystroke control. But once the last keystroke in the control file has been implemented, control reverts to the student (note that MECH ends the session at this point under normal keystroke control). Split control can be used to present information to students initially but then allow them to perform further exercises on their own afterward.

Prior to beginning keystroke control or split control, MECH presents an instruction file to students about how to control MECH during keystroke control. Once keystroke control ends, MECH presents another instruction file describing what students should do next (e.g., go on to self-control if split control is invoked, or wait for the instructor if the MECH session ends). The content of both instruction files is determined by the person authoring MECH's data files (see Section VII.F).

G. The Timed True-False Test

MECH provides the option of performing a timed true-false test at the end of a session. When initially configuring MECH, the instructor specifies that the test should be run and specifies the

IV. MECH as an Instructional Environment

As described in Section I, MECH can be used to provide people with conceptual knowledge about the structure of a domain and with practice on a variety of troubleshooting skills. Again we note that MECH is not a substitute for high-fidelity training situations. Instead, its optimal use may be in situations where only an abstract conception of a device is required, or in situations where conceptual training is integrated with high-fidelity training.

A. Learning Mental Models

The following subsections suggest possible ways MECH can be used to teach mental models to students.

1. Keystroke control. An instructor can use MOVE and TUTOR to construct a sequence of diagrams and descriptions that serve a particular instructional goal. Once the instructor exits MECH, the keystroke file saved from the session can later be "run" to direct students through the sequence of diagrams and descriptions viewed by the instructor. A large variety of "lessons" can be constructed in this manner.

For example, a "general lesson" that provides a rough idea of the domain could be constructed by only traversing the top-level systems of the domain. Later lessons could be constructed by traversing each of these systems in depth. From our configuration for small engines, a "general lesson" might only present the high-level diagrams and descriptions for the fuel system, ignition system, lubrication system, cooling system, cylinder assembly, and drive train. Six subsequent "specialized lessons" could then cover each of these high-level systems in depth. For example, the lesson on the fuel system would cover the fuel tank, fuel pick-up system, and carburetor.

The lessons described so far primarily follow the hierarchical structure of the domain, structuring lessons around the decomposition of systems. Alternatively, lessons can be organized along paths of functional relations. For example, a lesson could trace the path of fuel through an engine; another could trace the path of electricity; another could trace mechanical forces produced by the drive train.

Further flexibility can be achieved by manipulating the diagram and description files. Different lessons could use different diagram and/or description files that focus on different aspects of training. For example, students could be tutored on electronic circuits at two levels: a qualitative level and a physical level (White & Fredericksen, 1986). One set of diagrams and descriptions might present the qualitative structure of the domain; whereas another set might present physical mechanisms. Similarly, different sets of diagrams could be used to teach the structure of a device and procedures for repairing it. One set of diagrams and descriptions might present structure, whereas another set might present procedures.

2. User control. Rather than being under keystroke control, students could control MECH themselves during learning. An instructor could allow a student to browse MECH's diagrams and descriptions with no direction. Alternatively, instruction files could orient students toward

problem, jobs with the problem can be created, which the instructor solves. Subjects can first read about the problem in the instructions that start keystroke control and then watch the instructor solve instances of the problem. If split control is used, students can subsequently solve instances of the problem themselves.

Note that job descriptions can be used to teach students about problems. For example, various information in the job fields could provide students with information about engine models, engine ages, engine maintenance histories, and engine symptoms for common kinds of problems (see Section III.C.4 and VII.D.1). Even information about the actual faults and strategies for finding them could be included (e.g., in the OBSERVATIONS field). With a sufficient number of lessons, students could be exposed to the full range of problems known to occur for the system. In a sense, such training would develop subjects' ability to categorize symptoms into fault categories.

Fourth, lessons can be constructed to teach functional/qualitative reasoning. An instructor could first access a job description that specifies a non-working component and then run a test on that component. The message fired by the production controlling the test could state, first, that the component is malfunctioning and not broken, and second, that something in the path of the component must be malfunctioning. At this point, the instructor could use the FOLLOW INPUT option of the MOVE menu to trace inputs into the component, stopping at each previous component and testing it. In this way, the student could learn two important things about troubleshooting in situations where domain-independent strategies and symptom-fault rules are not optimal. First, the student could learn that it is important to reason functionally/qualitatively. By seeing the instructor do it across a number of problems, the student could learn the general strategy. Second, the student could learn specific functional paths within the system that might frequently be relevant to problem solving. For example, the instructor could trace the flow of fuel through the engine, the flow of electricity, or the flow of mechanical force. The instructions that start keystroke control could orient students to the strategy, and subsequent problems could allow students practice at using it.

Lessons on functional/qualitative reasoning could also be run in the opposite direction. The instructor could start with a broken component and then use the FOLLOW AN OUTPUT option in the MOVE menu to trace the effects of the broken component through the system. Or lessons could similarly address both forward and backward reasoning. The instructor could begin by tracing the causal effects of a broken component forward, running tests on components that receive faulty outputs from the broken component. Subsequently, the path could be run in reverse, showing students how to trace the reverse path of inputs during troubleshooting.

Finally, learning about mental models can be mixed with troubleshooting. An instructor could first present a problem. He or she could then use the MOVE and TUTOR utilities to teach students about the relevant part of the system. The instructor could then list relevant tests and repairs and then solve the problem. Learning mental models may be optimal under conditions in which students are also troubleshooting, and possibly vice versa.

V. MECH as a Research Environment

MECH has a number of characteristics that make it useful for conducting research:

(1) MECH is easily configured for a wide variety of experiments. An experimenter can vary instruction files, diagram files, description files, test-repair files, job files, help files, and keystroke control files. All these variations allow an experimenter to construct an indefinitely large class of independent variables.

(2) MECH collects rich information on dependent measures. MECH stores every keystroke plus its millisecond latency, along with complete information about the screen the subject was perceiving at the time (i.e., diagram and menu information). From this information, it is possible to reconstruct completely the conditions subjects experience and the physical operations they perform. These data can be used in a variety of ways to assess the effects of the independent variables in (1).

(3) MECH contains a timed true-false test that can be used to assess student's knowledge at the end of a session. Many other tests can also be used to measure subjects' knowledge at the end of a session, including free recall, cued recall, organizational tests, and ratings. We have not implemented any of these other tests in MECH so far and instead perform them with paper and pencil.

(4) MECH runs on IBM PC's, which are widely available as laboratory computers. As described in Section VIII.C, we plan to extend MECH to a wider range of processors such that it will be still more accessible to experimenters. One important extension would be to have MECH run on battery-operated lap-top computers that are highly transportable.

(5) MECH can be used to explore a wide range of issues on learning and troubleshooting. MECH can be easily configured to perform experiments on all the topics just covered in Section IV. We do not review these topics here. Instead we suggest that readers with experimental interests return to Section IV and reassess it from the perspective of an experimentalist. Later sections also suggest a variety of ways in which MECH can serve experimental goals.

2. The SOURCE 2 Disk

Root Directory

Filename	Description
move.inc	MOVE menu module (MECH)
params4.inc	parameters module for initialization (MECH)
readfalt.inc	processes broken/working conditions for utility4.pas (MECH)
readjobs.inc	reads job information for init4.pas (MECH)
repair.inc	REPAIR menu module (MECH)
test.inc	TEST menu module (MECH)
tutor.inc	TUTOR menu module (MECH)
video.inc	screen access utilities (MECH)

DIAGRAMS Directory

Filename	Description
diagcomb.com	combines binary screen images into MECH's diagram file
looker.com	displays binary screen image files
setrv.com	converts UPPERCASE fields in diagrams to inverse video
show.com	displays ASCII diagrams so that snapshot.com can capture them
snapshot.com	captures ASCII diagrams

ASCII Diagram	Binary Screen Image	Description
0000.asc	diagram.0	engine
1000.asc	diagram.1	fuel system
2000.asc	diagram.2	ignition system
3000.asc	diagram.3	lubrication system
4000.asc	diagram.4	cooling system
5000.asc	diagram.5	cylinder assembly
6000.asc	diagram.6	drive train
1200.asc	diagram.7	fuel tank
1300.asc	diagram.8	carburetor
1230.asc	diagram.9	fuel pick-up system
2100.asc	diagram.10	breaker points
2200.asc	diagram.11	condenser
2300.asc	diagram.12	magneto
2400.asc	diagram.13	spark plug
2500.asc	diagram.14	stop switch
2340.asc	diagram.15	coil
3300.asc	diagram.16	oil pump
6200.asc	diagram.17	crankshaft
6300.asc	diagram.18	camshaft

4. The DRIVE B Disk

Filename	Description
joblist.1	example of a job list file for small engines
joblist.2	example of a job list file for small engines
showkeys.exe	executable file for SHOWKEYS
11.flt	example of a problem fault file for small engines
12.flt	example of a problem fault file for small engines
13.flt	example of a problem fault file for small engines
14.flt	example of a problem fault file for small engines
15.flt	example of a problem fault file for small engines
16.flt	example of a problem fault file for small engines
21.flt	example of a problem fault file for small engines
jobs.hlp	example of a help file for JOBS (MECH)
move.hlp	example of a help file for MOVE (MECH)
repair.hlp	example of a help file for REPAIR (MECH)
test.hlp	example of a help file for TEST (MECH)
tutor.hlp	example of a help file for TUTOR (MECH)
kshelp1.ins	example of instructions for starting keystroke control
kshelp2.ins	example of instructions for ending keystroke control
ksstart.ins	example of instructions for starting keystroke control
ksstop.ins	example of instructions for stopping keystroke control
tfstart.ins	example of instructions for starting the timed true-false test
tfstop.ins	example of instructions for ending the timed true-false test
11.job	example of a job description file for small engines
12.job	example of a job description file for small engines
13.job	example of a job description file for small engines
14.job	example of a job description file for small engines
15.job	example of a job description file for small engines
16.job	example of a job description file for small engines
21.job	example of a job description file for small engines
kshelp.key	example of a keystroke data/control file
kshford.key	example of a keystroke data/control file
hford.par	example of parameter file
kshelp.par	example of parameter file
kshford.par	example of parameter file
tftest.tst	example of a true-false test for small engines

In general, if you have trouble getting MECH to run, you might check the config.sys and autoexec.bat files to see if they are configuring your system in a way that interferes with MECH. If you see something that looks suspicious, delete it (after saving it somewhere), reboot your computer, and try to run MECH. If MECH now runs, what you deleted is the problem. In this case, you might want to construct a boot disk specifically for using MECH, which does not contain your normal config.sys and/or autoexec.bat file. See Section VI.C.2 for one line you may need to include in your config.sys file.

C. Setting Up Disks and Drives

In setting up disks and drives, the following clusters of files *must not* be split. Files within these clusters *must not* be placed in different directories:

Cluster 1
All the files on the DRIVE A disk (MECH program files).

Cluster 2
All the parameter files (*.par) on the DRIVE B disk.

Cluster 3
All the help files (*.hlp), instruction files (*.ins), and keystroke control files (*.key) on the DRIVE B disk.

Cluster 4
All the job files (joblist.*, *.job, *.flt) and the true-false test (tftest.tst) on the DRIVE B disk.

Cluster 5
All the keystroke data files and all the true-false data files that MECH produces (i.e., *.key and *.tf). No such files currently exist on any of your disks (although the keystroke control files are technically keystroke data files).

The files within each cluster *must always* be in the same directory. For example, you could not put the HELP files from cluster 3 in one directory and the instruction files from cluster 3 in another directory. Note, however, that these clusters can be combined into larger clusters if so desired. For example, you could combine clusters 2, 3, 4, and 5 into a single directory if you wanted (and as we have done on the DRIVE B disk).

1. Running MECH from floppy disks. The first thing you should do is create back-up copies of the original MECH disks on four additional floppy disks. If you are using a machine with 1.2 MB floppies, you can backup up the original disks on a single disk.

When you want to run MECH, place the DRIVE A disk in drive A of your machine and place the DRIVE B disk in drive B. You are now ready to run MECH.

If you are using any other configuration of drives and disks, you will have to edit the respective parameter files before you can run the demonstrations. This is actually quite simple, as described in Section VI.F. If you are using some other configuration and want to run the demonstrations, read Section VI.F first. Then change the parameter files described in the demonstrations below to handle your configuration. This will simply amount to specifying the drives where MECH can find the files it uses for the demonstrations.

1. Demonstration 1: Overview. This demonstration first provides examples of the MOVE and TUTOR commands. It then accesses a job and repairs the engine. To run this demonstration, perform the following steps:

Step 1. Be sure that the root directory is the default directory on drives A and B.

Step 2. Log onto drive A.

Step 3. Type "mech".

Step 4. When prompted for the drive containing parameter files, type "b".

Step 5. When presented with a list of parameter files, type the *number* corresponding to the parameter file called "kshford.par".

Step 6. When the parameter screen appears, type "29".

Do not change anything in the parameter screen, unless you have read Section VI.F and are assigning new drives to fit your configuration.

Step 7. When the MECH logo appears (i.e., a spark plug at the top and the message "Please wait for the instructor" at the bottom), hold down the <ALT> key and type "200" *on the numeric key pad*. You *must* type the "200" on the numeric key while simultaneously holding down the <ALT> key. This obscure sequence is used to prevent students and subjects from controlling MECH at critical points (see Section VI.E. for details).

Step 8. MECH will take about 60 sec to initialize. Subsequently, you will receive a page of instructions, which will tell you how to proceed through the demonstration (these are the instructions to start keystroke control described earlier). Note that the demonstration is being controlled by a keystroke file, which we constructed during a previous MECH session. When the session is over, you will receive the instructions that end keystroke control.

Step 9. The MECH session will end automatically once you have completed the session.

2. Demonstration 2: The HELP utility. This demonstration takes you through MECH's HELP utility. If you want to learn about the MOVE, TUTOR, JOBS, TEST, and REPAIR

We use this obscure stop sequence so that students can't end MECH themselves, either intentionally or accidentally. As described in Section VI.H, this same key sequence also initiates the timed true-false test.

F. Setting Up Parameter Screens and Files

Once you start MECH, it asks you for the drive containing the parameter files. Enter the letter for the drive (see Section VI.C) followed by <ENTER>. Do not follow the letter with a colon (as occurs when setting the default drive in DOS).

MECH then produces a list of all the parameter files it can find on the drive you entered. To select a parameter file, type the number to the left of the filename, followed by <ENTER>. If you want to construct a completely new parameter file, select the number for that option.

MECH then presents the parameter screen. An example of a parameter screen is shown in Figure 3, where the parameter options are shown in UPPERCASE, and the values for these options shown are in lowercase. To set each option, simply enter its number when the prompt shown at the bottom of Figure 3 is on the screen. We next describe how to set each option.

1-CURRENT PARAMETER FILE. If you selected a parameter file when you initiated MECH, the name of this file will fill Field 1, and any values stored for this file will fill other fields on the screen. Note that there is actually a parameter file called "hford.par" on the DRIVE B disk. However, the values in this file are *not* the same as those shown in Figure 3.

If you selected the option to create a new parameter file, Field 1 and all other fields will be blank. Your first step in creating a new parameter file should be to enter a name for the parameter file. To do this, type "1" followed by <ENTER> to signal that you want to enter a value for Field 1. Then type the name of the new parameter file followed by <ENTER>.

2-NAME. This field contains the name of the person performing the MECH session. This field can be left blank, given it currently performs no function in MECH. As described in Section VIII.A, future versions will store this information in all data files.

3-DATE. This field contains the date of the MECH session. This field can be left blank, given it currently performs no function in MECH. As described in Section VIII.A, future versions will store this information in all data files.

4-CONDITION. If the person performing the MECH session is in an experimental condition, the name of this condition can be entered in this field. This field can be left blank, given it currently performs no function in MECH. As described in Section VIII.A, future versions will store this information in all data files.

5-NUMBER. If the person performing the MECH session has a subject number, it can be entered in this field. This field can be left blank, given it currently performs no function in MECH. As described in Section VIII.A, future versions will store this information in all data files.

6-RESULT FILE DRIVE. This drive contains the files in cluster 5 (see Section VI.C). Specifically, MECH will store the keystroke data file and the true-false data file from the current session in the directory assigned to this drive.

7-RESULT FILE PREFIX. The value for this field becomes the prefix of the keystroke data file and true-false data file that MECH produces for this session. MECH automatically assigns ".key" and ".tf" as the extensions for these files, respectively. For this reason, do not add an extension to the result file prefix, or MECH will not operate properly. Both result files will be stored on the drive specified in Field 6.

8-KEYSTROKE CONTROL? If all or part of this session should be under keystroke control, enter "y". If all of this session will be under user control, enter "n".

9-TRUE-FALSE TEST?. If the timed true-false test is to occur at the end of this session, enter "y". If not, enter "n".

10-PROBLEM FILE DRIVE. This drive contains the files in cluster 4 (see Section VI.C). Specifically, MECH will look for job files, fault files, joblist files, keystroke control files, and the true-false test file in the directory assigned to this drive.

11-PROBLEM FILE. This field contains the name of the file listing the jobs to be performed during troubleshooting (i.e., the joblist file described in Section VII.D.3). If troubleshooting will not be performed, this field must still contain a filename, or the program will not run.

12-OPTIMAL WAGE. This is the optimal amount that students can earn per problem. To see how it is used in computing a student's actual wage, see Section III.D. Note that the value for this field must include a decimal.

13-DELAY FACTOR. The value of this field determines the pause between each keystroke and subsequent screen. Note that you should enter an integer value and that it represents milliseconds. If you want the screen to change immediately following each keystroke command, enter "0" or some other small number (e.g., "30" works well). If you want the delay to be longer, enter a larger number (e.g., "2000" produces a delay of 2 seconds). Note that long values in the range of "2000" are necessary when using keystroke control. If the value is short, students can not see which options in the menus have been selected. When the values are long, students have time to see which option has been selected, thereby providing them with greater opportunity to learn.

28-SAVE THE CURRENT SETTINGS. If you have created a new parameter file, or if you have edited an old parameter file, select this option to save your work. The next time you initiate MECH, you can quickly reinstate these parameters by selecting the parameter file currently named in Field 1.

29-CONTINUE MECH. If you do not need to change any parameter, or if you are through editing parameters, select this option to begin a MECH session. If you edited parameters, you do *not* have to save them before continuing.

After you select this option, MECH will present the screen that contains its logo. At the bottom of this screen will be a message to wait for the instructor. When you are ready to continue the MECH session, hold down the <ALT> key and simultaneously type "200" on the number pad of your keyboard (see Section VI.E for details).

After you type <ALT> 200, MECH will spend about one minute initializing. If MECH is under user control, MECH will then present the 0000 diagram and the main menu. From this point on, you can select any option you wish. If MECH is under keystroke control, you will receive the instruction file named in Field 23.

30-EXIT MECH. If you do not wish to continue the MECH session, selecting this option will return you to the DOS prompt.

Important note about parameter files! You can not rename a parameter file with the DOS "rename" command and expect the parameter file to work. *All* creation, editing, renaming, and saving of parameter files *must* be done *within* MECH.

G. Using Keystroke Control

To use keystroke control, you must first create a keystroke data file under user control. After you have finished creating a keystroke data file under user control, you need to move the file to the directory that contains the files for cluster 4. You can then use this file to control MECH on any subsequent occasion. To do so, you must first set up the parameter file as just described in Section VI.F:

Step 1. If you wish to have complete keystroke control, Enter "y" in Field 8 of the parameter screen.

Step 2. If you wish to have split control, Enter "y" in Fields 8 and 14 of the parameter screen.

Step 3. Enter the name of the keystroke control file in Field 22 of the parameter screen.

Step 4. Specify names for the keystroke instruction files in Fields 23 and 24.

Step 4. Return to the MAIN menu (press F10 from whatever menu you are in).

Step 5. Once you have MAIN menu on the screen, hold down the <ALT> key and simultaneously type "200" on the number pad of your keyboard (see Section VI.E for details).

Step 6. The instructions named in Field 26 of the parameter screen will appear.

Step 7. Once the student has read these instructions, MECH initiates the true-false test. The instructions we have included in the tfstart.ins file on the DRIVE B disk describe how to perform MECH's timed true-false test.

Step 8. When MECH has finished the true-false test, it will save the results and present the instructions named in Field 27 of the parameter screen.

Step 9. When MECH has finished presenting these instructions, it returns to the DOS prompt.

Details on the data stored for the timed true-false test are described in Section VII.G.

I. Using MOVE, TUTOR, JOBS, TEST, REPAIR, and HELP

We do not describe how to use the MOVE, TUTOR, JOBS, TEST, and REPAIR utilities here. Everything you need to know about how to use these utilities is described in the HELP files we have constructed, namely, move.hlp, tutor.hlp, jobs.hlp, test.hlp, and repair.hlp, all of which reside on the DRIVE B disk.

You can read these files in several ways. First, you could run Demonstration 2 in Section VI.D.2, which presents all the HELP files under keystroke control. Second, you could run Demonstration 3 in Section VI.D.3 and explore the HELP utility under user control. Third, you could use the DOS "type" or "print" commands to review the contents of the HELP files on the DRIVE B disk.

J. Using SHOWKEYS to Read a Keystroke File

MECH stores its data files in a compact form to minimize storage space. As a result, these files are not very readable (although they are readable by data analysis programs). To make keystroke files readable to the human eye, we constructed SHOWKEYS.EXE, which resides on your DRIVE B disk.

To use SHOWKEYS, perform the following steps:

Step 1. Log onto the directory containing SHOWKEYS.EXE.

VII. Authoring or Altering MECH's Input Files

The instructions in this section allow you to:

(1) author entirely new files for a domain other than small engines
(2) alter files we have constructed for small engines

We generally assume throughout this section that the reader is interested in authoring new files. However, everything we say also applies to altering old files. If you are only interested in using the configuration of MECH that we have developed for small engines, and if you do not wish to change it in any way, then you can skip this section. If you plan to use our configuration, you may want to construct your own jobs for troubleshooting, your own instruction files, and your own true-false test. If so, then you should probably read subsections D, F, and G, respectively, in Section VII.

Most of the work in configuring MECH for a new domain involves authoring the files we describe in this section. More importantly, the success of a MECH application will probably rely to a large extent on how well these files are constructed. As described later in Section VIII.J, we hope to develop an authoring environment at a later time to facilitate this process.

Please note the following points about working with MECH input files:

(1) When constructing or altering files with a text editor, always be sure to save the files *unformatted*. If MECH finds any formatting commands in an input file, it will probably crash. If MECH crashes, you might check your files for formatting.

(2) Once you create or alter one of MECH's files, be sure to place it in the proper directory where MECH can find it. See Sections VI.C and VI.F for relevant instructions. If MECH can't find a file it needs, it tells you and then terminates.

(3) We do not specify the formats for files specifically. For example, we typically do not state exactly how many spaces must lie between adjacent fields in a line. Instead, we assume that authors will determine the appropriate formats from the example files we have constructed for MECH. The proper format is readily discernable from these examples (more discernable than if we tried to describe it).

A. Diagram File for MOVE

This lengthy section contains instructions for creating and editing MECH's diagram file. Because MECH's utilities for MOVE, TUTOR, TEST, REPAIR, and JOBS are hierarchically organized around a common addressing scheme, critical topics in this section include hierarchical organization and addressing. Sections 2 and 3 below describe how to construct such a scheme for your application.

Table 1
A Design Hierarchy and Addressing Scheme for Small Lawn Mower Engines

System/Terminal Component	Address	Diagram Order	External Relations Inputs	Outputs
ENGINE	0000	0		
FUEL SYSTEM	1000	1	O	
air filter	1100			
FUEL TANK	1200	7		P,Q
tank	1210			
gas cap	1220			
FUEL PICK-UP SYSTEM	1230	9		P
tank	1231			
fuel pipe	1232			
fuel pipe filter	1233			
fuel flow ball	1234			
metering holes	1235			
high-speed adj. screw	1236			
CARBURETOR	1300	8	P,Q	O
chamber	1310			
choke	1320			
venturi	1330			
throttle valve	1340			
low-speed adj. screw	1350			
IGNITION SYSTEM	2000	2	E,C	N
BREAKER POINTS	2100	10	E,I	L,M
plunger	2110			
moving arm	2120			
spring	2130			
moving point	2140			
stationary point	2150			
CONDENSER	2200	11	M	
housing	2210			
gasket	2220			
spring	2230			
tin foil	2240			
insulation	2250			
MAGNETO	2300	12	C,L	I,J,K
flywheel	2310			
magnet	2320			
armature	2330			
COIL	2340	15	L	I,J,K
housing	2341			
primary wire	2342			
secondary wire	2343			

(2) Each component at levels two through four can only be decomposed into nine sub-components each. In Table 1, "ENGINE" decomposes into six systems (e.g., FUEL SYSTEM, IGNITION SYSTEM). We could have decomposed "ENGINE" into three more systems at the most. Similarly, the largest number of components the carburetor could have had would have been nine (it actually had five).

As described in Section VIII.E, later implementations will not be constrained in these two ways.

It is important to note that one can be quite creative in designing hierarchies. For example, imagine you wanted to tutor people on electronic circuits at two levels: a qualitative level and a physical level (White & Fredericksen, 1986). The root of your hierarchy could have two branches, one that presents qualitative structure and one that presents physical mechanisms. As another example, imagine you want to train people about three devices. The root of your hierarchy could have three branches, one that presents a subhierarchy for each device. As still another example, imagine you wanted to teach people about the structure of a device and procedures for repairing it. The root of your hierarchy could have two branches, one that presents structure and one that presents procedures.

Alternatively, you could construct completely separate hierarchies for different purposes. Instead of only using one hierarchy, you would construct several, each of which could be stored in a different diagram file for use in different sessions. For example, you could load the hierarchy for a qualitative level in one session and load the hierarchy for a physical level in another session.

Step 2: Determine internal relations. Once you have constructed the hierarchical decomposition of your domain, you need to determine the internal relations within each decomposable system (in Table 1, decomposable systems are shown in UPPERCASE). Figure 2 in Section III.C.2 provides examples of internal relations. In the ignition system diagram, for example, the magneto sends high voltage current to the sparkplug. In the magneto diagram, movement of the magnet produces magnetic reversal in the coil.

At this point, it may be useful to sketch out each diagram, including its components and the internal relations between them. You may want to draw your diagrams along the lines of those shown in Figure 2. Or you may want to draw them in some other way.

Step 3: Determine external relations. For each diagram you sketch, identify the following two kinds of external relations: (1) *inputs* the system receives from other parts of the system, and (2) *outputs* the system sends to the rest of the system. All the external relations that we use in our current implementation are shown in Table 1. Each relation needs to have a label, which *must* be a capital letter (as described in Section VIII.A, this constraint will be removed in later versions). Consider an example from Table 1. P,

(1) The maximum length of a diagram file is 18 lines. This is because the menu system uses the remaining lines at the bottom of the screen. The number of lines in a diagram can be less than 18.

(2) The maximum length of any single line within a diagram file is 79 characters.

(3) DO NOT name any of your ASCII files "diagram". The binary screen images created from these files in Section VII.A.6 will be given file names of "diagram.#" by the SNAPSHOT.COM program. Using this name for any ASCII file will create unwanted problems.

(4) We suggest that you name your ASCII diagram files as follows: Use the address of the diagram (from your addressing scheme) for the prefix of the filename. For the suffix, use "asc" to remind yourself that these are ASCII files. For examples of such filenames, see the names for the ASCII files in Section VI.A.2 (e.g., 0000.asc, 1000.asc, 6200.asc)

(5) Within each diagram, the name of any component that decomposes should be in UPPERCASE. In Figure 2, for example, the magneto in the ignition system diagram decomposes into the diagram for the magneto below. Consequently, MAGNETO is in UPPERCASE in the ignition system diagram. As described in Section VII.A.7, UPPERCASE names in diagrams will appear in inverse video. This makes them more salient in the diagrams, thereby making it clearer to subjects which components decompose and which do not.

Note that *extended* ASCII characters can be used to construct lines, corners, t-junctions, arrows, shading, etc. in your diagram files. Use of the extended character set can produce surprisingly good diagrams. The trick is finding a text editor that will allow you to enter them into a file. Microsoft WORD allows a user to enter extended ASCII characters by conjoining the use of the <ALT> key and the numeric key pad. For examples of how ASCII characters can be used to construct diagrams, see the diagrams we constructed for our current configuration of MECH.

As described in Section VIII.D, we plan to add bit-mapped graphics to MECH in the future. Once we have done that, the basic routine we describe in Section VII.A for constructing diagram files should be much the same. The only difference would be that diagrams are constructed with a graphics editor instead of with a text editor.

Once you have constructed your ASCII diagrams, your working directory should contain the five utility files described in VII.A.1 and the ASCII diagrams you created with your text editor.

5. Order the diagrams. Next you must calculate the order of the diagrams for your configuration of MECH. To see how we ordered the diagrams for small engines, see the "Diagram Order" column in Table 1. You may be able to discern the ordering principle more easily from seeing how diagrams are ordered in Table 1 than from the more abstract procedure we describe shortly.

decomposable systems that have not been added, namely, 1200 and 1300. If the systems from 1000 have already been added, assign the current node to be 2000, assuming its decomposable systems have not been added yet. If they have, continue on across the descendents of 0000, seeing if any have decomposable systems that have not been added. Go to Step 1.

Step 2b. If the answer is "no" and the current node is 0000, you are done. If the answer is "no" and the current node is anything but 0000, then change the current node to the superordinate of the current node. For example, if the current node is 1200, assign its superordinate, 1000, to be the new current node. Go to Step 1.

As you construct the diagram order, write the position of each diagram next to its address. This information will be necessary in later steps. For an example, see Table 1.

6. Create binary screen images. The next step is to create a binary screen image for each of the ASCII diagrams that you created in Section VII.A.4. In creating these binary screen images you will use the SHOW.COM, SNAPSHOT.COM, and SETRV.COM programs that reside in your working directory.

Step 1: Load SNAPSHOT.COM into memory by simply typing "snapshot" at the DOS prompt. Because SNAPSHOT is a memory resident program, you only need to run it once during a session in which you create binary screen images.

Step 2: This step is iterative, being repeated once for each ASCII diagram that you want to convert to a binary screen image. For example, if you want to convert 19 ASCII diagrams to binary screen images, you would repeat this step 19 times.

Most importantly, you *must* perform these conversions according to the diagram order you just constructed in Section VII.A.5. You must start with the first diagram in the order, and continue with the remaining diagrams until the last diagram in the order has been converted. If you do not convert diagrams in the correct order, MECH will not run properly.

To convert each ASCII diagram, type "show *filename*", where *filename* is the full name of the ASCII file that you wish to convert to a binary screen image. For example, type "show 0000.asc" to convert the ASCII file for the engine to a binary screen image. This displays the file on the screen. Note that when the diagram is displayed, no DOS prompt will be present. Once the diagram has been displayed, SHOW will pause, waiting for further input from you. At this point, press the <CTRL> and <BREAK> keys simultaneously. This will cause SNAPSHOT, which is resident in memory, to create a binary screen image of the displayed ASCII diagram.

Repeat the complete process outlined in this step until all ASCII diagrams have been converted to binary screen images. As SNAPSHOT creates each new binary screen image, it automatically assigns a filename to the new image. The first binary screen image

(1) The conversion utilities.
(2) The ASCII diagrams you created with your text editor.
(3) The binary screen images, with inverse video, that you created with SHOW, SNAPSHOT, and SETRV.

8. Create a single diagram file. The binary screen images must now be combined into a single data file for use by MECH. This is done through the use of the DIAGCOMB.COM utility, as outlined next.

Step 1: At the DOS prompt, type "diagcomb" followed by <ENTER>.

Step 2: When DIAGCOMB's menu appears, select option "c" to implement combination.

Step 3: This is an iterative step that must be performed once for each diagram. After you press "c" in the previous step, diagram.0 will appear on the screen, and a counter will appear below it. To incorporate diagram.0 into the final diagram file, press <ENTER>. At this point, diagram.1 will appear on the screen. To incorporate this diagram into the final diagram file, press <ENTER>. Continue with this sequence until all diagrams have been incorporated into the final diagram file

Step 4: Once combination has finish, select the "e" to end the session.

Your working directory will now contain a file called "diagram.dat". This is the file that MECH uses to present diagrams. Note that the name of the diagram file MECH uses *must* be called "diagram.dat." As described in Section VIII.A, the name of this file will be a variable in future implementations. Be sure to place the "diagram.dat" file in the directory that contains your cluster 1 files (see Section VI.C)

9. Editing diagrams. If you decide to change a particular diagram once you have created the diagram.dat file, first edit the ASCII file for the diagram. Then convert it to a binary screen image, using SHOW and SNAPSHOT, and then convert its UPPERCASE strings to inverse video, using SETRV. Finally, run DIAGCOMB to integrate your edited diagram into a new diagram.dat file.

10. Viewing screen images. Whereas DIAGCOMB allows you to view binary screen images within diagram.dat, the LOOKER.COM utility allows you to view individual binary screen images.

Step 1. At the DOS prompt, type "looker" followed by <ENTER>.

Step 2. Once LOOKER has been loaded, it will prompt you for a screen mode. Select option two for 80 column text.

Table 2
Example of an Entry for a Decomposable System in the tutor.dat File

DIAGRAM = lubrication system
ADDRESS = 3000
SPECIFIC DESCRIPTIONS = 1 2 3
NODES BELOW = 3
DECOMPOSITIONS = 3
INPUTS = F-6300 R-6000
OUTPUTS = S-5000 T-6000
GENERAL DESCRIPTION = LUBRICATION SYSTEM
The lubrication system minimizes friction and wear on engine components, thereby maintaining optimum operating conditions. At one point in the rotation of the camshaft, oil is pulled into the oil pump through suction. At a subsequent point in rotation, oil is forced out of the oil pump and into the intake port. Oil is then forced out of the port injector in the form of a mist. This mist falls onto the internal surface of the cylinder assembly and onto the drive train to keep them lubricated.

1 = INTAKE PORT
The intake port is a small area at the end of the oil line just before the port injector. It collects oil that is to be forced out of the port injector and onto the cylinder assembly and drive train areas.

2 = PORT INJECTOR
The port injector is a small nozzle on the side of the intake port. Because of the nozzle's small opening, oil forced out of it is turned into a mist. Consequently the port injector turns the oil into a form suitable for distribution over the internal surface of the cylinder assembly and over the drive train.

3 = OIL PUMP
The oil pump transfers oil from the crankcase to the intake port. Rotation of the drive train causes oil to be pulled into the pump from the crankcase, where oil is stored. Further drive train rotation forces oil out of the pump and into the intake port.

specific description (see the SPECIFIC DESCRIPTIONS field). Consequently, the lubrication system has specific descriptions for the intake port, port injector, and oil pump.

Important note! The description just described can contain no blank lines. When a blank line is encountered, MECH assumes that the description has ended.

2. Create entries for terminal components. Table 3 shows an example of an entry for a terminal component. As can be seen, only four fields need to be specified:

DIAGRAM must contain "none", because terminal components do not have their own diagrams.

ADDRESS contains the address of the terminal component.

NODES BELOW is 0, because this is a terminal component.

GENERAL DESCRIPTION contains the name of the terminal component on the same line.

Note that no general description of the terminal component follows. This is because the specific description for this component exists in the entry for its superordinate. For example, the specific description for the intake port is in the entry for the lubrication system (see Table 2).

3. Order the entries. Organizing entries is similar to organizing diagrams, as discused in Section VII.A.5. To see how we ordered the entries for small engines, see the tutor.dat file on the DRIVE A disk. You may be able to discern the ordering principle more easily from seeing how entries are ordered in tutor.dat than from the more abstract procedure we describe shortly.

As can be seen in tutor.dat, the entry for engine is first, followed by entries for the six systems that compose it at the next level down (fuel, ignition, lubrication, cooling, cylinder, drive train). The next entries are for the components of the fuel system, beginning with an entry for the gas cap, which is a terminal component. The next two entries are for the fuel tank and carburetor, thereby exhausting the immediate descendents of the fuel system. The next entries are for the immediate descendents of the fuel tank, followed by the immediate descendents of the fuel pick-up system, and finally by the immediate descendents for the carburetor. At this point, entries for the ignition system begin to be added.

More precisely, the general procedure for ordering entries is as follows:

Step 0. Assign 0000 to be the "current node." Also create 0000 as the first entry in tutor.dat.

Step 1. For the current node, determine if entries for its immediate descendents have been placed in tutor.dat. For example, if the current node is 0000, determine if entries for 1000, 2000, 3000, 4000, 5000, and 6000 have been added.

Step 1a. If the answer is "no," create an entry for each immediate descendent, in numerical order. If an immediate descendent is decomposable, enter information as described in Section VII.B.1 above. If an immediate descendent is a terminal component, enter information as described in Section VII.B.2 above. For example, if the current node is 0000, first add 1000, then 2000, then 3000, then 4000, then 5000, and then 6000. Exhaustively add all components that are immediate descendents before continuing to Step 2.

Step 1b. If the answer is "yes," then go to Step 2.

Step 2. Determine if any entries remain to be added from the immediate descendents of the current node *at any level*. For example, if the current node is 0000, determine if any entries have not been added for descendents of 1000, 2000, 3000, 4000, 5000, or 6000 at any level.

Step 2a. If the answer is "yes," change the current node to the first decomposable descendent having entries that remain to be added. For example, from 0000, assign the current node to be 1000, because it is the first decomposable descendent having entries that remain to be added, namely, 1100, 1200 and 1300. If all components from 1000 at all levels have been added, assign the current node to be 2000, assuming it has descendents with entries that remain to be added. If all the entries for 2000 have been added, continue on across the descendents of 0000, seeing if any have entries have not been added. Go to Step 1.

Step 2b. If the answer is "no" and the current node is 0000, you are done. If the answer is "no" and the current node is anything but 0000, then change the new current node to the superordinate of the current node. For example, if the current node is 1200, assign its superordinate, 1000, to be the new current node. Go to Step 1.

It is essential that you save the tutor.dat file *unformatted*. If your text editor adds formatting commands to the file, MECH will not be able to read it properly.

C. Description File for TEST and REPAIR

This next file contains the tests and repairs used by TEST and REPAIR. The name of this file must be "testrep.dat". As described in Section VIII.A, the name of this file will be a variable in future implementations. A fragment of the testrep.dat file that we constructed for small engines is shown in Table 4.

There are eight constraints on designing and ordering entries in testrep.dat:

(1) The maximum number of tests/repairs is three per component. As described later in Section VIII.E, we plan to remove this constraint in future implementations.

(2) Tests and repairs are yoked: For every test in testrep.dat, there must also be a repair (or lack of repair, as described next).

(3) There may be no repair for a test. In general, there may be no repairs for tests of high-level systems because the repairs must take place at the level of more specific components. Nevertheless, a description of some sort *must* be included for these "non-repairs" in entries where they occur (e.g., see repair #1 for the carburetor).

(4) A cost must be included for each test and repair. These are used to calculate payoffs to subjects. If there is no repair for a test, the repair receives a cost of $ .00.

(5) Tests specify whether the engine needs to be "on", "off", or "either" when they are performed. Repairs similarly specify whether the engine needs to be "on", "off", or "either", assuming a repair exists. If a repair does not exist, it receives a value of "none" for the engine's on/off status (e.g., see repair #1 for the carburetor).

(6) The description of a test can be up to 2 lines long, with each line containing a maximum of 72 characters and ending with a return. The description of a repair can also be up to 2 lines long, with each line containing a maximum of 72 characters and ending with a return. As described in Section VIII.E, we plan to remove the constraints on number of lines in future implementations.

(7) Each line of information in an entry must be followed by a line containing an "*" to separate it from the next line of information.

(8) Entries in the testrep.dat file must be ordered according to the procedure described for tutor.dat in Section VII.B.3.

To see the basic format for constructing entries in testrep.dat, consider the tests and repairs for the carburetor in Table 4. More specifically, each entry should be structured as follows:

Line 1 contains the following three pieces of information:
(1) the address of the component (1300)
(2) the number of yoked tests/repairs (2)
(3) the name of the component (carburetor).

Line 3 contains the following five pieces of information:
(1) the number of the test/repair (#1)
(2) the cost of the test (dcost=$ 4.00); the "d" stands for "diagnostic"
(3) whether the engine must be "on", "off", or "either" to run the test ("off")
(4) the cost of the repair (rcost=$ 000.00)

Table 5
Examples of Job Files (21.job, 11.job)

```
PROBLEM = 21
NAME = fuel system tune up, adjust high- and low-speed adjustment scre
IDEAL COST = $ 16.00
*
CUSTOMER = 1
Owned by the Ann Arbor Parks Department.
MODEL = 1
Made by Tecumseh. Heavy-duty model.
AGE = 1
About 1 year old.
USE = 1
Used five days a week to cut lawns in city parks.
MAINTENANCE = 1
Has received no maintenance in 1 year.
CURRENT PROBLEM = 2
Engine stalls when running at low speeds.
Engine doesn't go fast enough at high speeds.
PREVIOUS PROBLEM = 1
None.
OBSERVATIONS = 1
Operator thinks that the fuel system might be out of adjustment.


PROBLEM = 11
NAME = malfunction at spark plug -> broken secondary wire (explicit #1)
IDEAL COST = $ 32.00
*
CUSTOMER = 1
Owned by the Sea Palms Apartment Complex.
MODEL = 1
Made by Craftsman.  Made in the United States.  Discount model.
AGE = 1
About 1 year old.
USE = 0
MAINTENANCE = 0
CURRENT PROBLEM = 1
The engine will not start.
PREVIOUS PROBLEM = 0
OBSERVATIONS = 2
A mechanic friend of the customer checked out the engine a little and
found that the spark plug is not producing a spark.
```

field will not appear in any form when MECH runs. From the student's perspective, these fields will not exist.

As described in Section VIII.A, later versions of MECH will allow job files to contain optional fields of any description, not just these eight.

Each optional field is defined by a first line that (a) defines the field, and (b) specifies how many lines will subsequently constitute its value. Consider the field for CURRENT PROBLEM in Job 21. "CURRENT PROBLEM=2" defines the field and specifies that two lines for it follow. In Job 11, when "0" follows the name of a field, this means that neither the field nor the value will appear in MECH's job descriptions. Only those fields and values having at least one line are shown.

The value for each field can contain any number of lines. Each line must be a maximum of 79 characters and end with a return.

2. Fault files. An example of a fault file is shown in Table 6. This is the file, 21.flt, on your DRIVE B disk. Additional examples of fault files reside on the DRIVE B disk as well.

The following six constraints apply to constructing a fault file:

(1) Analogous to job files, the maximum number of fault files is 1000.

(2) The prefix of each filename must always be a number from 0 to 999. The extension of each filename must always be ".flt". Examples of filenames include 11.flt, 21.flt, 142.flt, etc.

(3) The prefix of each fault file should be the same as the prefix of the corresponding job file. For example, 11.job and 11.flt form a pair, as do 21.job and 21.flt. If you do not use identical prefixes in this manner, MECH will not operate properly.

(4) The first five lines shown in Table 6 *must* begin every fault file. The subsequent lines are optional, as described in Sections VII.D.2.a and VII.D.2.b below.

(5) Line 1 contains the number of faults for the current problem. A fault corresponds to a particular repair of a particular component that must be made before the problem is solved. A problem can have any number of faults.

(6) Line 2 contains the specific repairs that must be made to eliminate the *n* faults introduced in Line 1. Each repair is indexed by:

(a) the address of the component that needs to be repaired
(b) the number of the repair for that component that needs to be performed.

For example, "1236 2" means that the high-speed adjustment screw in the fuel-pick up system must be opened to repair the first fault (i.e., test #2 of component 1236). "1350 2" means that the low-speed adjustment screw in the carburetor must be opened to repair the second fault (i.e., test #2 of component 1350). Line 3 must contain an "*".

As each needed repair is made, MECH "erases" that repair from what is currently wrong with the system. This changes the productions that fire when the student starts the system and runs tests, as described in Sections VII.D.2.a and VII.D.2.b below.

Important note! If a component is duplicated in your system hierarchy, you probably should not assign it as a fault during troubleshooting. Consider the flywheel, which is duplicated several times in our hierarchy for small engines. If the flywheel is at fault, one or all of the nodes for the flywheel could be specified as faults. If only one is specified, then tests on different nodes for the flywheel would yield different results, which is obviously inappropriate. Worse yet, if all flywheel nodes are specified as faults, then the flywheel would have to be repaired several times before the problem was solved. This, too, is clearly inappropriate.

One way to handle this problem would be to designate one node as the one that subjects test and repair. For all other nodes, the Test #1 and Test #2 descriptions would state that tests and repairs could not be done for the current diagram. In addition, these descriptions would direct the user to the one diagram where the tests and repairs could be performed. For example, the Test #1 message could state, "The flywheel can not be tested here. Go to the crankshaft diagram to test the flywheel." This is how we have handled repetition for the flywheel and (fuel) tank in our current configuration for small engines.

(7) Line 4 must contain the number of test messages, where each test message is a production. The number of productions (test messages) is unlimited. There are two different types of productions:

(a) "System productions" that apply to starting the system (described in Section VII.D.2.a below).

(b) "Test productions" that apply to running particular tests on particular components (described in Section VII.D.2.b below).

(8) Line 5 must contain an "*".

a. System productions. System productions apply to starting the system. Certain tests and repairs may require that a student start the system before performing them (see Sections III.C.5 and III.C.6). Furthermore, every time a student makes a repair, MECH requires that the

(c) Line 2 specifies the "working conditions" of the production. If the production is to fire, it's working conditions must match the current state of the engine. In other words, the production must find that the tests specified by these conditions do not reveal faults. To do this, it simply looks up the current list of faults for the engine. The maximum number and format of working conditions are the same as for broken conditions.

In general, either working or broken conditions *alone* are sufficient to specify the conditions of a production fully. We have included both so that users have more flexibility in designing problems.

(d) Line 3 and Line 4 (if used) specify the "action" or message that the production produces. If the production's working *and* broken conditions are met fully, the production "fires" by presenting its message to the student. A message can contain a maximum of two lines, each containing a maximum of 79 characters and ending by a return. As described in Section VIII.E, the constraint on number of lines will be removed in future implementations.

b. Test productions. Test productions apply to performing tests. Every time a student attempts to run a test, MECH searches for productions that apply to the test. If one matches the current fault state of the system (as just described in Section VII.D.2.a), then it fires its message. If no productions fire, then MECH assumes that the test found nothing wrong and responds with a default message stating that the test found no fault.

A general rule of thumb is that test productions need to be written for :

(a) Components whose tests exhibit faults (e.g., the last two productions in Table 6).

(b) Components whose tests are affected by existing faults (e.g., the productions for 1000 2 and for 1200 2 in Table 6). Note that 1000 2 and 1200 2 are not faults themselves. Instead the fuel system (1000) and the carburetor (1200) are malfunctioning because of other faults that exist in the system.

Knowing which productions of type (b) to construct is the trickiest part of authoring MECH files. In our examples, we simply reasoned intuitively about faults to produce these productions. For example, we assumed that improperly adjusted low- and high-speed adjustment screws would affect the carburetor and fuel system and that these faults would affect no other test in the system. In some cases, producing productions in this intuitive manner may be sufficient for constructing an application.

Other applications, however, may require that test productions be designed more rigorously. You may need to consult an expert or a text to determine which non-broken components of a system are affected by a particular fault. As we discuss in Section VIII.L, if an actual simulation of a system exists, it can be used to define these productions. Moreover, it should be possible to

Table 7
Example of a Joblist File (joblist.1)

```
21
11
12
15
16
```

Table 8 presents fragments of the true-false test we constructed. If you wish to browse through the rest of the test, see the tftest.tst file on the DRIVE B disk.

The first line of the file must specify the total number of probes. Each subsequent set of three lines corresponds to a probe item. The format for these items is as follows:

Line 1 contains a code for each item, taking up the first eight character fields of the line. Note that blank spaces in these fields count as 0's. The contents of these eight character fields are defined as follows:

(a) The first three characters fields of the code specify the item's number. Because the first three characters for the first item in Table 8 are " 1", this means that the item's number is "001."

(b) The fourth character field indicates whether the item is true or false. As can be seen from Table 8, "1" indicates that the correct response is "true," and "2" indicates that the correct response is "false."

(3) The fifth character field specifies the type of item. As can be seen from Table 8, "1" items test components, "2" items test internal relations (i.e., relations shown in *some* diagram), "3" items test relations not shown in *any* diagram that must be inferred, and "4" item test relations that involve the environment.

(4) The sixth character field specifies the level of the probe in the diagram hierarchy. As can be seen from Table 8, "1" items occur at the level immediately below 0000, "2" items occur one level lower, etc. For the "3" and "4" items just described above for (3), the sixth character field specifies the hierarchical level of the *lowest occurring* component in the relation.

(5) The seventh character field is always blank.

(6) The eighth character field only applies to the "3" and "4" items just described above for (3). For these items, the eighth character field specifies the hierarchical level of the ***highest occurring*** component in the relation.

Line 2 and **Line 3** for each item contain the information presented to subjects. Each of these two lines can be a maximum of 79 characters and must end with a return. If an item only requires one line, the second line must nevertheless be left blank.

Important note! You can define fields 4-8 in whatever way you want. You do not need to define them as we have defined them. However, fields 1-3 must be used to represent the number of the probe.

The test can contain up to 212 items. As described in Section VIII.E, this limit will be removed in future versions of MECH. Items must be ordered by number, as defined by the first three

characters of each line (see (a) above). When MECH presents the test to a student, it presents the probes in a random order.

Note that MECH stores the following information for each item in the true-false data file after running the true false test:

(1) The item number (characters 1-3 of the item code).
(2) The remainder of the eight-character item code.
(3) The subject's response (true or false).
(4) The subject's confidence in their response (1 = low, 2 = moderate, 3 = high).
(5) The subject's reaction time in milliseconds.

MECH does not compute any results based on fields four through eight of the item code. Instead this information is stored for use by external data analysis programs. MECH stores the data for probes according to their order in the file from which they were read. MECH does not currently store the position of the probe in the test sequence. As described in Section VIII.A future versions will also record this information.

(5) In the JOB INFORMATION menu that lists the characteristics of previous jobs, we need to include the faults that were repaired. These could simply be read out of the fault file that accompanies the job file.

(6) In the JOB INFORMATION menu that lists the characteristics of previous jobs, the menu option "F10 Return to prev menu" should be changed to "F10 Return to previous menu".

(7) A number of options need to be added the parameter screen. Later sections (referenced below in parentheses) describe these options in greater detail. Adding all these options may require a second "page" in the parameter screen, where each page can accessed by a "switch page" option in the other.

(a) Add options for files whose names are currently constants in the MECH program. In particular, add options for:

(a) The name of the diagram file used by MOVE (curently diagram.dat).
(b) The name of the description file used by TUTOR (currently tutor.dat).
(c) The name of the description file used by TEST and REPAIR (currently testrep.dat).
(d) The name of the logo screen (currently nameplat.dat).

(b) Add an option for the file containing menus and messages (see Section VIII.B).

(c) Add three options for:

(a) Type of diagram (ASCII or bit-mapped).
(b) Screen driver (standard monochrome, Hercules monochrome, CGA, EGA, or VGA).
(c) Menu control (keyboard, mouse, or both).

See Section VIII.D for discussion of these additions.

(d) Add an option for the filename of the COMMENT file (see Section VIII.F).

(e) Add an option that asks whether a video disk is being used (see Section VIII.K).

(f) Add an option that asks whether a qualitative simulation is being used (see Section VIII.L).

(8) Options for filenames need to be blocked by cluster in the parameter screen, as defined in Section VI.C. Each block should be preceded directly by the option for the drive that contains the cluster below.

(17) In Section VI.J we discussed a problem in using SHOWKEYS, namely, that MECH requires the original parameter file to be unchanged. To remove this problem, and to remove having to access the parameter file altogether, MECH should extract whatever it needs from the parameter file and store it with the data files when they are originally created.

B. Implementing Complete Domain-Independence

We originally thought we had implemented complete domain independence in MECH, only to discover that a few domain-dependent features remain in the menus and messages of the core program and in the description fields of job files. For example, domain-dependent menu options include "Start the engine" and "Stop the engine". Domain-dependent messages include "The engine will not start". Domain-dependent job fields include "MODEL" and "MAINTENANCE". If someone wanted to configure MECH for an electronic circuit, these options, messages, and fields would not be suitable.

What we plan to do is make the contents of menus, messages, and job fields be *file-dependent.* Rather than being part of the source code, the contents of menus, messages, and job fields would exist in MECH's input files. Although users could not change the structure of the menus, the location of the messages, or the processing of job fields, they could change the names of menus, the names of menu options, the contents of messages, and the names of job fields.

For example, users could edit the "surface aspects" of menus and messages in a menu/message input file, much like those constructed for MOVE and TUTOR. The name of this file would be an option in the parameter screen. Upon initializing, MECH would incorporate the contents of this file into its configuration for the current session.

For job fields, MECH could simply read the job fields in the file for each job, where each field is indicated by a preceding @ (see Section VIII.A, item 8).

Important note! If a user wants to configure MECH for a domain, but does not want to reprogram MECH to handle menus, messages, and job fields flexibly, then menus, messages, and job fields could be changed easily by editing the source files for MECH. Such changes would only require retyping those character strings that constitute menu names, menu options, messages, and job fields. Once these changes had been made, MECH would have to be recompiled.

C. Implement Extensions to Other Processors

MECH currently runs on a limited set of computers (see Section VI.B.1). Whatever aspects of MECH limit it to these machines must be generalized such that MECH can run on any computer using the DOS operating system (e.g., IBM, Compaq, Zenith, AST) or processor (e.g., 8088, 8086, 80282, 80386). We also hope to develop a version that runs in the APPLE environment.

(5) A maximum of 36 working conditions and 36 broken conditions for productions in fault files.
(6) A maximum of two lines for the message of a production in fault files.
(7) A maximum of 1000 jobs in a joblist file.

(8) A maximum of 212 probes on the true-false test.

All other relevant aspects of MECH are unconstrained in size, as far as we know.

F. Add the COMMENT UTILITY

We need a general utility that can present a file of information at any point during a MECH session. For example, this utility could be included in keystroke control to present instructions, strategies, etc. at various points during learning. Or this utility could be triggered to provide hints when students under user control experience difficulty while troubleshooting.

More specifically, the COMMENT utility should be a procedure that creates and implements productions of the following form:

Condition: An eliciting keystroke sequence or event.
Action: Presentation of a file.

There appear to be two primary ways such productions could be constructed, one for keystroke control and another for user control. For keystroke control, the eliciting condition could be a fixed keystroke sequence, such as "<ALT> 100". Whenever an instructor enters "<ALT> 100" under user control, MECH would:

(1) Ask for the name of the file that should be presented at this point later during keystroke control.
(2) Record the filename.
(3) Ask whether the full screen should be used to present the file, or whether the file should be presented in the lower menu area below the diagram.
(4) Store a call to COMMENT in the keystroke file that specifies the information in (2) and (3) as arguments.

Subsequently, when the keystroke file is used for keystroke control, encountering this call to COMMENT will automatically present the student with the file. Note that the student should not see steps (1) - (4) above. Instead MECH should simply present the file at the appropriate time and at the appropriate place on the screen. As in all our other text presentations, MECH should allow the user to scroll through the file if it is larger than the window in which it is presented.

COMMENT could be utilized under user control for troubleshooting as follows. Another option could be added to the parameter screen for a "comment file". In this file would be

point, MECH would show the map in the diagram part of the screen and state where the user is in the lower part of the screen.

This conception of the MAP utility is actually quite simplistic. It really doesn't add much information to our current use of paths on the upper-left corners of menus (as described in Section III.B). It should be possible, however, to conceive of more powerful versions of this utility. If users find that our current use of paths is sufficient, the MAP utility may not be necessary.

H. Add the SEARCH Utility

Imagine that someone wants to use MECH for online support during actual troubleshooting. One way to provide such support would be for the user to access a SEARCH option from the MAIN menu. The SEARCH menu would allow three options.

(1) The user enters symptoms from a real job into a register.

(2) The user receives a known list of symptoms in alphabetical order and enters those into the register that describe symptoms in the current problem. The list would be the union of symptoms from *all* job files in the current configuration of MECH (such a list could be quite long).

This menu option would be useful when the symptoms entered in (1) above don't retrieve any cases in (3) below. This could occur if a user does not describe symptoms in the same terms that describe symptoms in the job files.

(3) The user could use whatever symptoms were placed in the register by (1) and/or (2) to search all the pairs of job and fault files in the current configuration of MECH. The symptoms in the register could be combined by a Boolean "and" or "or" to form different types of probes for search. Any job matching the current probe would be retrieved.

As jobs matching the symptoms are retrieved, the user could browse through them, looking at their faults and/or other characteristics. This information could provide hypotheses about what the fault(s) might be in the current problem.

Besides adding the SEARCH routine to MECH, it would be necessary for authors to construct large sets of job and fault files that represent the variety of problems that occur in a domain. The problems could represent real or fictional jobs. To the extent it is possible to identify a fairly exhaustive set of problems, and to the extent they are all represented in job/fault files, MECH could serve as a means to access this data base in ways that might be useful during real troubleshooting.

The SEARCH utility might also be useful in training students to troubleshoot. Rather than solving problems when given symptoms, students could look up known cases when given

show an arrow pointing to that component. For example, if a user wanted to see where the venturi is in the carburetor, the video disk could now show the carburetor with an arrow pointing at the venturi.

Integrating conceptual and visual information in this manner may be a particularly powerful way of training students about a system, because they learn about it at multiple levels. Students would simultaneously learn about the hierarchical organization of components, the functional organization of components, and how to identify real instances of components.

(2) Video footage could be associated with every address in the testrep.dat file. New options in the TEST and REPAIR menus could then allow users to see how a given test or repair is actually performed. For example, a user could actually watch someone testing the ratio of air to fuel in the air-fuel mixture. Or a user could actually watch someone replacing the spark plug. In this way, students could acquire some degree of procedural knowledge.

Again, integrating conceptual and perceptual training may be ideal. From our current MECH utilities, students learn domain-independent strategies, symptom-fault rules, and functional/qualitative reasoning. From the video disk, students could learn how to perform these skills.

(3) When students access information about jobs, the video disk could show pictures of the actual device that needs to be repaired and the problematic symptoms it exhibits. Associating this visual information with the more abstract information currently provided by MECH would greatly enhance the categorization skills that underlie fault diagnosis.

If video disk technology were added to MECH, an option for using or not using it would have to be added to the parameter screen.

L. Add an Interface for Qualitative Simulations

Currently, an author has to write productions for every problem. As described in Section VII.D.2, the basis for these productions could lie in intuition, be obtained from experts, or be obtained indirectly from a simulation. However, MECH could be interfaced directly to a simulation, thereby allowing the following functions:

(1) An instructor could "break" part of the simulation, thereby creating jobs (i.e., fault files would contain information about which part of the simulation to break).

(2) When initially presented with a job, a user could try to start the simulation and acquire symptom information about what is not working properly.

References

Collins, A., Salter, W., & Tenney, Y. (1987). Contract progress report. Bolt, Beranek, and Newman. Cambridge, MA.

Hegerty, M., Just, M.A., & Morrison, I.R. (1988). Mental models of mechanical systems: Individual differences in qualitative and quantitative reasoning. *Cognitive Psychology, 20,* 191-236.

Johnson-Laird, P.N. (1983). *Mental models.* Cambridge, MA: Harvard University Press.

Murphy, G.L., & Medin, D.L. (1985). The role of theories in conceptual coherence. *Psychological Review, 92,* 289-316.

Norman, D.A., & Shallice, T, (1986). Attention to action: Willed and automatic control of behavior. In R.J. Davidson, G.E. Schwartz, & D. Shapiro (Eds.), *Consciousness and self-regulation: Advances in research and theory* (Vol. 4). New York: Plenum Press.

Schank, R.C., Collins, G.C., & Hunter, L.E. (1986). Transcending inductive category formation in learning. *The Behavioral and Brain Sciences, 9,* 639-651.

White, B.Y., & Frederiksen, J.R. (1986). Intelligent tutoring systems based upon qualitative model evolutions. In the proceedings of *AAAI-86: The national conference on artificial intelligence*. Philadelphia.

Appendix C
The First Round of Follow-up Experimentation
Excerpts from Interim Report
ARI Contract No. MDA903-90-K-0112

The psychology group has performed many experiments in the first year of this project, investigating the different types of knowledge used in troubleshooting and the effects of each on learning and assessing the role of third-party explanations and self-explanations in troubleshooting and learning how to troubleshoot.

0.1 The role of device topography and causal knowledge in troubleshooting

The earliest experimentation was on the role of device topography and causal knowledge in troubleshooting. Of interest were the roles that these two factors play in people's ability to acquire symptom-fault categories. Subjects first learned information about a system that they were going to troubleshoot (i.e., a satellite refueler or a glass brick maker, both fictional). All 96 subjects in the experiment learned the components of a given system. Half of these subjects also learned the system's hierarchical topography (which was isomorphic for the satellite refueler and glass brick maker). Subjects learned the system's components plus topography until they had memorized them. Subjects then performed troubleshooting. On each trial, subjects received the name of a component in the system that was malfunctioning, and their task was to select from a list of other components the component that was the cause of the malfunction. In a sense, this is a categorization task, because subjects must categorize each symptom into a fault category. We manipulated the relation between the symptom and the fault. For subjects who learned the topography, the symptom either occurred before or after the fault in the topography (direction), crossed with the symptom being either close or far to the fault (distance). The symptom and fault always occurred within the same subsystem in the hierarchical topography. All experiments are constructed with subjects via computer and take around 2 hours per subject. Subjects participate either for course credit or pay. To induce subjects to perform well, we pay all subjects according to how efficiently they perform troubleshooting. For subjects who are being paid to be in the experiment, this bonus is added on top of their basic wage.

Three robust effects demonstrated the effects of topography on troubleshooting. First, subjects preferred to select faults from the same subsystem as the symptom. Second, subjects preferred to select faults that occurred before the symptom in the topography. Third, subjects preferred to select faults that were close to the symptom within the same subsystem. All three effects clearly indicate that subjects use topography to construct hypotheses about the cause of a symptom. Subjects who did not learn topography showed no such biases. A fourth effect, unexpected but quite interesting, was that subjects were better at troubleshooting the satellite refueler than at troubleshooting the glass brick maker. This is

surprising, because both systems have the same topography. The effect suggests that causal knowledge about the relations between particular subsystems and components also affects subjects' troubleshooting hypotheses. Because the causal processes are different for these two systems, differences in troubleshooting result.

In another experiment, we explored the role of causal processing futher. In this experiment, the topography of the system was maintained, but the components were scrambled within it, thereby precluding any causal understanding of the system. If causal processing is a factor, then troubleshooting should be worse than in the unscrambled condition. We have added an additional condition that also maintains the topography but only has letters indicating the components, thereby again precluding causal understanding. These conditions were compared to a standard, unscrambled topography group, along with a group who learned no topography.

All subjects learned the same system components (26 of them) and the same abstract topography prior to learning diagnosis (a two level hierarchy, with one level of 6 components arranged in a branching graph, with 4 of these components decomposing into 4 chains of 5 components each). But whereas half of the subjects received the components in their normal positions in the original topography (coherent topography), the other half received the same components placed randomly within the topography (scrambled topography). In other words, both groups received the same components and the same abstract pattern of connectivity between components, but one group received the components inserted in their normal meaningful positions, whereas the other group received the components inserted randomly into positions. If subjects simply use patterns of connectivity (i.e., topography) to learn symptom-fault pairs, then this manipulation should not affect learning in the diagnosis phase of the experiment. But if subjects further interrelate symptoms and faults according to causal knowledge, scrambled subjects should experience difficulty and perform poorer than standard topography subjects. Not only was this prediction upheld, scrambled subjects performed more poorly than no topography subjects (i.e., subjects who only learned the components in random orders and learned no topography; see our prior progress report). Consequently, subjects use both topography and causal knowledge to learn symptom fault pairs.

A third experiment addresses the role of topography in learning fault categories whose symptoms vary across problems. In the prior two experiments, only one symptom ever occurs for a given fault category. These are categories in the limit, because only one exemplar occurs per category. In this third experiment, we create variability among category exemplars. Again, we manipulate whether subjects know a topography, and we look for subsystem and distance effects (not direction effects this time). Each fault category is defined by the presence of one critical symptom. But in some conditions, irrelevant, non-predictive symptoms accompany the critical, predictive symptom. Whereas the critical symptom stays constant across trials, the irrelevant symptoms vary from trial to trial. Across conditions, we manipulate whether there are 0, 1, or 2 irrelevant symptoms associated with the members of a fault category.

We expect to see performance drop as the number of irrelevant symptoms increases, because it becomes increasingly hard for subjects to isolate the critical symptom. However, we expect much less of a drop for subjects who know topography than for subjects who do not. The reason is that the critical symptom is always close to the fault and ahead of it in the same subsystem, whereas the irrelevant symptoms are always far away in other subsystems. If subjects know the topography, they can use it to isolate the critical symptom, such that irrelevant features have less of an effect.

Given we found substantial effects of topography in our previous two experiments, we were quite surprised to find none in this experiment. In trying to explain this unexpected result, we developed several hypotheses concerning conditions that produce topography effects. We have designed a series of three experiments to test these hypotheses. However, these experiments are on the back burner while we address more pressing issues raised by the first two experiments.

0.2 Why does topography help?

In presenting results from the first two experiments to our colleagues, we discovered that we had glossed over a critical issue. Topography subjects could perform better for either of two reasons or both:

1. Topography enables sophisticated guessing about where faults are likely to lie. As described in the previous progress report, faults in our experiments are likely to lie in the same subsystem as a symptom, to lie after the symptom in a temporal sequence, and to lie close to the symptom within the subsystem. If subjects know the topography, they can restrict search considerably, thereby finding faults faster.

2. Topography enables subjects to develop stronger associative relations between symptoms and faults. If subjects have knowledge about connectivity and causal processes, they can use it to understand why a fault produces a symptom, and they can develop a more robust memory structure that promotes faster and more permanent learning.

Both of these factors are central to diagnosis, potentially having significant implications for learning and performance. However, our previous analyses of Experiments 1 and 2 did not distinguish which of these two factors produced the topography effects we observed. Consequently, we developed a simple math model of learning, which models percent correct during diagnosis as a function of a parameter for learning rate and several parameters for various sources of guessing. Because we know the percent correct from our data, we can estimate values for learning rate, once we establish all of the guessing parameters. Of interest is whether the guessing parameters and learning rates differ across conditions. By assessing these differences, we can determine which of the above two factors are controlling

performance. Nearly all of the data necessary for assessing this model existed already in Experiments 1 and 2. However, we also needed to collect scaling data for one additional guessing parameter that is essential to the model.

Upon applying the model to our data, we found that topography only affected guessing. Much to our surprise, topography did not affect learning. Although our central prediction had been that topography would facilitate learning, and although our preliminary results had indicated this to be true, these more analytic results show convincingly that topography is only affecting guessing. Still more surprisingly, we found that the distribution of symptom and faults, not only facilitated guessing, but also facilicated learning (we had originally predicted that a predictable distribution would only affect guessing). When the distribution of symptoms and faults across problems enabled subjects to restrict their search for faults, they learned symptom-fault rules faster, as estimated by our model.

To corroborate these surprising findings, we ran two further experiments that test for possible differences in learning rates more directly. Both used a new dependent measure, trials to criterion, which is a more direct index of learning rate. Essentially, these experiments are designed to rule out any effect of guessing. For subjects to succeed in diagnosis, they must learn the associations between symptoms and faults. If different conditions produce different learning rates, then we should see large differences in performance that correspond directly to this measure.

Rather than estimating learning indirectly through our model, we wanted to assess it more directly from an observable measure. In the first experiment, we manipulated the presence versus absence of topography, as well as the number of symptom-fault rules being learned (4 or 8). In the second experiment, we manipulated the presence or absence of topography distribution, together with the distribution of symptom-fault relations (i.e., the extent to which subjects could use their background knowledge to restrict search).

The paradigm we used to assess learning rates directly is similar to our previous one in that subjects learn different knowledge about the system prior to diagnosis (e.g., coherent topography, scrambled topography, no topography). Subsequently, subjects learn symptom-fault pairs during diagnosis. The key difference between these two experiments and the previous three experiments concerns the nature of the test during diagnosis: In our previous studies, subjects received a symptom followed by a set of possible faults in a forced choice recognition test. Subjects continued to select a possible fault from the forced choice list until identified the correct fault. In these new experiments, subjects generate what they believe is the correct fault–they do not select the fault from a forced choice list. After generating what they believe is the correct fault, subjects receive feedback about whether their response was correct and then go on to the next problem. To measure learning rate, we set the following criterion: Subjects meet the learning criterion for a particular problem once they generate its fault correctly three times in a row. The critical measures include the average trial at which criterion is reached, the average ratio of correct to correct-plus-incorrect trials, the average first correct trial, the average number of correct trials prior to reaching criterion,

and so forth. Each of these measures tells us something about the rate at which subjects are learning associations between symptoms and faults. Guessing is not a factor, because it is extremely unlikely that subjects could guess the correct response three times in a row. To meet the criterion, subjects must have learned the symptom-fault association.

In piloting this study, we found the largest differences between conditions that we have observed so far. Specifically, coherent topography produces faster learning than no topography, which produces faster learning than scrambled topography. We are also looking at learning rates in terms of direction and distance (e.g., are learning rates faster for backward than for forward faults; are learning rates faster for close faults than for far faults). In many ways we are quite pleased with this new paradigm; it replicates our previous results using a fairly different learning task; it seems to produce stronger and cleaner results; and it provides direct measures of learning rate uncontaminated by guessing.

In the first of these two experiments (our fourth experiment overall), we also manipulated the number of symptom-fault pairs, with half of the subjects learning 4 symptom-fault pairs (as in our previous experiments) and the other half learning 8 pairs. As in our previous experiments, subjects receive all of their 4 or 8 problems in a random order in each of many blocks (12 in the current experiment). We expected to see this manipulation magnify our topography effects: The more problems that subjects have to manage and learn, the more topography helps.

In the second of these two experiments (our fifth experiment overall), we manipulated the distribution of problems over subsystems. In the previous four experiments, the symptom for every problem came from the same subsystem in the topography as the fault. This biased distribution of problems is what enables topography subjects to have an advantage at guessing faults (in Experiments 1 and 2). In this fifth experiment, we wanted to see if distribution underlies the advantage in learning rate that the topography condition exhibits over the other conditions. To assess this issue, we had one third of the subjects again receive problems whose symptom and fault always lie in the same subsystem. In contrast, for another third of the subjects, the symptom and fault always lie in different subsystems. Finally, for the remaining third of the subjects, the symptom and fault lie in the same subsystem half of the time and in different subsystems the other half of the time. If distribution matters, then it should moderate the topography effects. For example, topography effects might disappear for problems that cross subsystems, because subjects now have trouble establishing mediating relations based on connectivity and causal processes. Alternatively, there may still be a topography advantage, if topography subjects use the topography as a reference system for coding the symptom and fault in a pair, regardless of whether they come from the same subsystem or not.

As in the first two experiments, we found no effect of topography on learning rate. Also, as in the first two experiments, we found an effect of a predictable distribution on learning rate.

In trying to explain the surprising effect of distribution on learning (as well as on guessing), we arrived at the following model: As the number of potential faults increases, the strength of the strongest incorrect response increases (following a basic statistical property of distributions). Consequently, the correct response requires higher strength to exceed its nearest competitor, as the set of possible faults increases. If subjects can reduce the scope of search, the number of competitors decreases, thereby requiring less strength for the correct response to exhibit learning.

This role of search size on learning strikes us as a novel discovery. The importance of search in memory and problem solving and memory is well known. But typically it's effects are assumed to occur prior to learning, when an intelligent system has no idea of what the correct response might be. Under these conditions, a system is searching for a novel response it has never made before, and the problem is the number of incorrect responses that must be checked initially. In contrast, our model assumes that the correct response has already been learned through previous events and that it is examined on every trial. The problem is not that the subject must search through many irrelevant responses before finding the correct one. Instead, the problem is the strength of the correct response is weak initially and susceptable to interference from other faults. As the number of other faults considered decreases, there is less chance that an incorrect response will win the competition.

Consequently, our account views search in a fairly novel way, much like the relation of signal to noise in signal detection theory: As the noise increases, it gets harder to detect the signal. Learning doesn't simply amount to the strengthening of responses. Instead, learning also involves discovering constraints on search that make it easier for the accumlated strength of the correct response to exceed the noise and control processing.

However, our colleagues have pointed out alternative explanations that also account for out data. First, subjects may be more likely to construct explanations between symptoms and responses when they occur predictably near each other. It could be that the presence of explanations for predictable faults versus the absence of explanations for unpredictable faults accounts for our distribution effect. Second, subjects may be following Luce's choice rule in selecting a fault, where the probability of a particular fault is the ratio of its strength to the sum of the strengths for all faults in the search set. As the number of faults in the search set increases, they decrease the relative probability of the correct fault.

We have begun designing experiments to discriminate among these possibilities. For example, we can assess the explanation hypothesis by making faults predictable but never having them occur in the same subsystem as the fault. Because this view assumes that explanations are easiest to construct within the same subsystem, it predicts that performance should be worse when symptoms and faults occur in different subsystems as compared to when they occur in the same subsystem (holding predictability and therefore size of the search set constant). On the other hand, our competition view predicts that these two conditions do not differ, because the size of the search set does not change.

We can also assess the explanation hypothesis by seeing if the strength of symptom-fault relations is weaker when search sets are small than when they are large. Because of less competition for small search sets, relations don't need to be as strong to exhibit criterial learning, according to our model. In contrast, the explanation hypothesis still predicts that relations should be stronger for small search sets, because it's easier to construct explanations under these conditions.

To assess the Luce choice hypothesis, we can hold the strength of the nearest competitor constant at intermediate strength while varying the number of weaker responses. Because our max hypotheses states that only the strength of the nearest competitor matters, it should not matter how many weaker alternative exist in the search set. In contrast, the Luce choice hypotheses assumes that the probability taken away from the correct response increases with the number of incorrect responses, no matter how weak they are.

Currently, these experiments to distinguish explanations of our distribution effect are being designed and are on hold, but we do expect to continue work on them in the next year. These investigations are tied in closely with the AI work supported by this funding. Although we have failed to find effects of topography thus far on learning (we have only found them on guessing), we still believe that topography affects learning. The problem is that our previous experiments probably do not induce subjects to use explanations as much as they might.

This brings us to our third line of experimentation, discussed in the next section: assessing the role of explanation in troubleshooting and learning to troubleshoot. However, we believe that Experiments 1, 2, 4 and 5 constitute a package that provides compelling and informative results about the role of topography in diagnosis. Making these results clear and presenting them to the community is a major goal in year two of this project.

0.3 Incorporation of third-party explanations into understanding a system

The third line of research in psychology begins to ask questions that bear directly on the AI work funded by this project. Specifically, we are interested in how people incorporate explanations provided by an external source into their understanding of a system, how they use this incorporated information in diagnosis, and how various factors remind people of this information later.

There are a long series of experiments that must be performed to explore this area, and we have only begun the series. Under what conditions is explanation most effective – when subjects infer their own explanations, or when they are provided with explanations at test time (i.e., after they have attempted to solve the problem)? Are third-party explanations better when given while the student is solving a problem or after they have come up with their own best solution? What types of explanations are there, and which kinds of explanations

work best for teaching which kinds of knowledge? Three particular kinds of explanations we investigate are functional explanations, topographical explanations, and causal explanations. From this set of studies, we should learn a lot about the effects of explanation on learning symptom-fault associations, as well as about how particular types of explanations affect learning at particular times.

In two initial studies, we simply looked at whether explanations produce faster learning during diagnosis. As in the previous studies, subjects learn various types of information about a system prior to learning diagnosis (i.e., coherent topography, scrambled topography, no topography). In the first of these studies, half of the subjects learn diagnosis exactly as described for Experiments 4 and 5. The other half produce an explanation linking each symptom and fault every time they solve a problem. Of interest is how generating explanations affects the acquisition of symptom-fault associations. We expected to see a general benefit due to producing explanations (based on a variety of well-known phenomena in the memory literature). Of more interest, however, was whether generating explanations completely overrode all of the previous effects. Do explanations compensate for scrambled topography or no topography? Do they eliminate direction and distance effects? We control for learning time by presenting symptoms and faults for the same amount of time in both conditions (i.e., in the explanation condition, producing explanations will add time to the task).

Three issues were of interest: First, do explanations provided to people while they are learning a system's structure facilitate the later learning of symptom-fault associations? Second, do explanations provided to people while they are learning symptom- fault associations (after they have learned the system's structure) facilitate learning? Third, are functional explanations maximally effective in facilitating learning, or does added causal information improve learning further?

We ran approximately 60 subjects in the two experiments that address the first two of the above issues, and much to our surprise, we observed little if any beneficial effect of explanations of any kind at any point in learning (in comparison to control groups who received no explanations). Consequently, we stopped performing these experiments and tried to understand why the explanations weren't working.

To understand the hypothesis we developed, consider a few details of how these experiments work. During the initial learning of a system, subjects study diagrams of the system's structure, with each diagram representing a set of physical components connected by their input-output relations. For example, one of the diagrams for the satellite refueler might represent several chemical tanks, which are connected to a mixer that blends the chemicals to form fuel. In this diagram, subjects would learn that chemicals flow from the chemical tanks to the mixer, where they are mixed. Later, during symptom-fault learning, subjects might learn that when the mixer malfunctions, it is typically because a chemical tank is broken. Consequently, the symptom is a malfunctioning mixer, and the fault is a broken chemical tank. In the experiments, subjects receive "mixer malfunctioning" as the symptom, and "chemical tanks broken" as the fault. The way to think about this kind of troubleshooting is

a real world situation where the troubleshooter is dealing with a modular system, for which various monitoring and diagnosis systems isolate malfunctioning and broken components. For example, a monitoring system might initially warn the operator that the mixer is malfunctioning. The operator might then run various diagnosis procedures, which specify that a chemical tank is broken, causing the mixer to malfunction. At that point, a repair person might swap out the malfunctioning tank with a replacement.

Note that in the above malfunction and diagnosis, no information was provided about the behavior of the malfunction (i.e., how the mixer was malfunctioning), or about what was wrong with the chemical tanks. Instead, the operator simply receives information that the mixer is malfunctioning, and that a chemical tank is broken. Our hypothesis about why explanations weren't helping subjects learn symptom-fault rules was as follows: Perhaps subjects couldn't access the explanations from the symptoms, because the symptoms were too "barren," not containing any behavioral information that would be specific enough to retrieve the explanations. For example, imagine that the symptom is "malfunctioning mixer," and that the explanation for why the a broken chemical tank caused this malfunction is that "the mixer is malfunctioning because, a chemical tank is failing to send a necessary chemical, thereby causing the mixer to produce incorrect fuel mixture." Because the symptom ("malfunctioning mixer") fails to specify its behavioral problem ("producing incorrect fuel mixture"), the learner has difficulty accessing the explanation that contains this information. In contrast, when the symptom is elaborated as in, "mixer that's producing an incorrect fuel mixture," the match is higher, and the explanation is more likely to be retrieved, thereby providing access to the probable fault. To summarize, perhaps we weren't seeing explanation effects, either at learning or test, because, the symptoms weren't specific enough to retrieve the explanations.

To test this, we ran a few pilot subjects for whom we provided behavioral information about the malfunctioning component in the symptom, and we finally found explanation effects. As a result, we redesigned the four original experiments to incorporate this critical variable, namely, whether or not the symptom is elaborated behaviorally or not. In these experiments, this factor is crossed fully with two additional factors: (1) whether subjects receive explanations at learning, and (2) whether subjects receive explanations at test. Note that explanations at learning describe how the system works as subjects study and memorize the diagrams for the system.

The implications of this finding appear important for designing troubleshooting applications in the real world. Essentially, it suggests that whatever symptom information subjects receive should be specific and should map closely onto the explanation that links it to the associated fault. For example, many modular systems nowadays simply instruct the operator that a particular component is malfunctioning, without specifying the malfunctioning behavior. If the operator knows an explanation linking the symptom to the fault, it is likely that this barren error message will not activate the explanation and thereby fail to take advantage of the explanation's ability to generate the fault. In contrast, if the error message states the behavioral properties of the symptom in a way that maps closely onto the

explanation, the explanation is likely to become active and bring the fault to the operator's attention.

We have also begun a thorough analysis of the explanations subjects form as they are solving problems. We are doing this through a protocol study. Subjects go through the basic learning and test phases of our standard experiment. But rather than simply collecting measurements on how well they have learned the system or on how well they have learned symptom fault rules, we collect online protocols of what subjects are thinking throughout the experiment. During learning, subjects explain how they think the system works. During testing, subjects explain why they believe certain faults produce certain symptoms.

We collected online explanations that our subjects produced as they learned, first, the structure of the system, and second, the symptom-fault rules. In contrast, to the previous experiments, we did not provide subjects with explanations but instead observed the ones they generated. Of interest is the types of explanations that subjects generate and how these explanations evolve over the course of learning.

We have finished piloting this study, and the results are quite promising. Subjects construct different kinds of explanations under different conditions that illuminate our previous work and provide many hypotheses for future work. We expect to report more in this area next year.

We are also planning further studies that assess the role of remindings in using explanations. Of particular interest are the conditions that produce the best remindings of previous explanations. If subjects can retrieve prior explanation when they are useful to solving a current problem, we will see benefits in problem solving. This series of experiments, too, builds on findings of the AI modeling work.

Appendix D
Learning by Understanding Explanations
Selected Papers

Redmond, M. and Martin, J. (1988). Learning by Understanding Explanations. In *Proceedings of the 26th Annual Southeast Regional ACM Conference*, pp. 524 – 529.

Martin, J. and Redmond, M. (1988). The Use of Explanations for Completing and Correcting Causal Models. In *Proceedings of the 10th Annual Conference of the Cognitive Science Society.*

Martin, J. and Redmond, M. (1989). Acquiring Knowledge by Explaining Observed Problem Solving. *ACM SIGART Newsletter*, No. 108, pp. 77 – 83.

In Proceedings of the 26th Annual Southeast Regional ACM Conference

Learning by Understanding Explanations*

Michael Redmond
Joel Martin

School of Information & Computer Science
Artificial Intelligence Research Group
Georgia Institute of Technology
Atlanta, GA 30332-0280
(404) 894-5550
E-mail: redmond@gatech.edu, joel@gatech.edu

Abstract

When complex systems fail, a great deal of domain specific knowledge must be used to diagnose the underlying fault. If a diagnosis system is to improve its performance through experience, efficient methods of knowledge acquisition and knowledge use are required. An approach to this problem is the Explanation-Based Learning (EBL) paradigm in which a causal model is operationalized for efficient use. However, the EBL approach does not allow for the addition of new domain knowledge, only the manipulation of existing knowledge. "Learning by Understanding Explanations" is a paradigm in which such learning can occur. In this paradigm, the system has a mental model of the domain that might have been derived from textbooks or tutoring, just as in the case of a human student. The system or student would attend to an instructor as that instructor solved an example case and would try to understand each statement or action. When the instructor explains something that is not yet understood, the system debugs its mental model, adding or changing information. Understanding an explanation, in general, will require inferences about that explanation in the context of what the system already understands and knows about the example case. Therefore, the current state of the system's knowledge will determine how the explanation is understood and will, in part, determine what is learned. Integration of the explanation into the working model includes adding knowledge implied by the explanation, correcting existing contradictory information, and forming diagnostic short cuts, called symptom fault sets, which associate a given fault with possible hypotheses. The novel characteristic of our approach is its ability to use new knowledge that is provided by an instructor to bridge gaps in explanations while debugging previous information in the mental model. We have experimented with two preliminary implementations of this approach using two different knowledge representations.

1 Introduction

Diagnosis of malfunctions in complex systems requires a great deal of domain specific knowledge. A frequent means of instruction in many diagnostic domains is the presentation of cases for the students to solve, followed by the instructor solving the same cases. This is true in wide ranging areas such as automobile mechanics, medicine, and management strategy.

In human protocol studies, Lancaster and Kolodner [LK87] examined the evolution from novice to expert in auto-mechanics students. From these studies has arisen the paradigm "Learning By Understanding Explanations" (LBUE). In this paradigm, the students have a mental model of cars and other useful information which they have developed from book knowledge, instruction and experience. The mental model includes among other things, causal relationships. The students follow along with the instructor, making inferences from what he does. When the instructor explains something that deviates from their understanding, they debug their model, adding or changing information. In addition the students can generate diagnostic short cuts, called symptom fault sets, which associate a given fault with possible hypotheses. This paradigm allows the addition of new causal knowledge, whereas classic Explanation-Based (EB) (EBL [DeJ83] [DM86] , and EBG [MKK86]) techniques cannot.

A general model of this learning paradigm and two preliminary implementations have been developed. The novel characteristic of the present approach is its ability to use new knowledge that is provided by an instructor in filling a gap in an explanation while debugging previous information in the mental model. The paper will present the general concept and the two implementations.

2 General Model

2.1 Introduction

The current research approaches the problem of learning to diagnose by examining what is learned from an instructor's hypotheses and subsequent explanations. Conceptually, this process involves understanding why a symptom occurs, why a particular hypothesis is proposed, and why a particular explanation is given.

When a system is attending to an expert's diagnosis, it must make inferences from the given information to fill in the omitted information. One way to do this is causal chaining. When it receives a symptom, it builds backward chains to all possible findings that could lead to the symptom. When it sees a hypothesis, it builds forward chains to all possible findings that could be caused by the hypothesized fault. If forward and backward chains meet, then the student has an explanation for the hypothesis. It can collapse the chain into its fault and symptom, and save it in a symptom fault set or in the mental model. The collapsed chain could then be used more efficiently in future, similar situations.

*This research was supported by the Army Research Institute for the Behavioral and Social Sciences under Contract No. MDA-903-86-C-173. The authors wish to thank Janet Kolodner for her advice and guidance, and Roy Turner and Mark Graves for helpful comments on earlier versions of the paper.

Often, however, the student will not have enough information to explain the hypothesis. Then the instructor's explanation can be useful. If the explanation is *X causes Y*, it can be added directly to the mental model with high credibility. It also may allow bridging a gap to complete a causal chain, thus enabling EBG by adding to the causal domain knowledge. If it does not do so, the system still can infer that the hypothesized fault causes *X*, and that *Y* causes the symptom, though there may be intermediate causal links.

As has been noted by many researchers [MKK86], EB methods do not generate any knowledge that the system does not already have; existing knowledge is reorganized to be more useful. In collapsing the chains, the LBUE method has some similarities to Explanation-Based Generalization (EBG) [MKK86] and Explanation-Based Learning (EBL) [DM86]. The student has used available knowledge to form an explanation, which is then stored, somewhat like a macro operator in STRIPS [FHN72], for later use. The LBUE concept goes beyond explanation-based methods by being able to use new information to create a chain when it would otherwise not be possible.

2.2 Algorithm

1. From the symptom, chain backward toward possible findings
2. From each hypothesis, chain forward toward possible effects
3. If the symptom chain meets a hypothesis chain, then the generalization that *(cause hypothesis symptom)* is added to the symptom fault table and to the mental model
4. If the chains do not meet
   (a) Chain backwards from the explanation toward the hypotheses
   (b) Chain forward from the explanation toward the symptom
   (c) If both directions can be linked, then the most general relationship *(cause hypothesis symptom)* can be learned
5. Add explanation to the mental model

2.3 Inputs and Outputs

A system implementing this paradigm should be given a current mental model of the domain, a set of likely symptom-fault pairings, and the representation of an expert's diagnostic episode. Following Lancaster and Kolodner [LK87], the mental model contains knowledge structures for components and processes, with each structure representing information about inputs, outputs, and structural, functional, and causal relationships with other structures. The structures are similar to frames [Min75], with the above knowledge filling appropriate slots. In one of the implementations, the similarity to frames is more functional than structural. A general example of part of the mental model is shown in Figure 1. Other relationships can be expressed through levels of abstraction. An initial version was described in more detail in Allison [All87].

The symptom-fault pairings were suggested by Lancaster and Kolodner [LK87], who observed their existence in human protocols. These sets of pairings associate a symptom with a problem, and are used to derive initial hypotheses during diagnosis, and to index into the mental model at the appropriate place. An example of what a generalized symptom-fault set looks like is given in Figure 2.

Since natural language processing is not the focus of this research, the diagnostic episodes are input as a list of clauses, representing symptom, hypotheses, tests, explanations, or information (Figure 3).

```
Starter:
        (isa component)
             ;A starter is a component.
        (part-of starting-system)
             ;A starter is a part of the starting system.
        (input electricity battery battery-cables)
             ;A starter receives electricity from
             ;the battery via battery cables.
        (parts starter-pinnion-gear starter-motor)
             ;A starter has parts: pinnion gear and
             ;starter motor.
        (function spin-action starter-pinnion-gear)
             ;The function of the starter is to
             ;spin the pinnion gear.
        (cause (switch-action solenoid on)
               (adjacent starter-pinion-gear
                         flywheel-ring-gear))
             ;Setting the solenoid switch causes
             ;the pinion gear and the flywheel gear
             ;to become adjacent.
        (cause (crank starter-pinion-gear)
               (crank flywheel-ring-gear))
             ;Cranking the pinion gear causes the
             ;flywheel gear to crank.
```

Figure 1: A Generalized Mental Model Definition of a Starter.

```
Generalized Symptom-Fault Sets
Symptom: ((NOT (START)) CAR)
 Faults:((MALFUNCTION FUEL-SYSTEM), credibility = 3)
        ((MALFUNCTION STARTING-SYSTEM), credibility = 1)
        ((MALFUNCTION AIR-INTAKE-SYSTEM), credibility = 1)

Symptom: (CRANK ENGINE-SYSTEM SLOW)
 Faults: ((MALFUNCTION STARTING-SYSTEM), credibility = 4)
         ((MALFUNCTION CHARGING-SYSTEM), credibility = 2)

Symptom: (MALFUNCTION FUEL-SYSTEM):
 Faults: ((MALFUNCTION FUEL-LINE), credibility = 4)
         ((MALFUNCTION FUEL-TANK), credibility = 2)
         ((MALFUNCTION FUEL-PUMP), credibility = 1)
```

Figure 2: Generalized Symptom/Fault Sets. 'Credibility' represents the likelihood that a particular pair holds.

The output is an updated mental model and a cognitive trace of the learning process. The following things may be learned:

1. new objects
2. new relationships between objects
3. new causal information
4. new symptom fault knowledge

```
(defvar "protocol@13") ; Made up protocol
(setq "protocol@13" '((SYM ((Not (Crank)) Engine-system))
    (HYP H1 ((Not (Contains)) Fuel-Tank Gasoline))
    (TEST (Contains Fuel-Tank Gasoline) (Type Visual)
                (Result ((Not (Contains)) Fuel-Tank Gasoline)))
    (HYP H2 (Corroded Battery-Terminals))
    (EXP H2 (Frequency (Corroded Battery-Terminals) High) )
    (EXP H2 (Cause (Corroded Battery-Terminals) ((Not (Connected))
Battery Electrical-wire)))
    (TEST (Corroded Battery-Terminals) (Type Visual)
  (Result (Corroded Battery-Terminals)))
    (ACTION (Scrape Battery-Terminals Corrosion))
    (TEST (Corroded Battery-Terminals) (Type Visual)
  (Result ((Not (Corroded)) Battery-Terminals)))
    (HYP H3 (Connected Battery-Cables Ground Loose))
    (TEST (Connected Battery-Cables Ground) (Type Visual)
  (Result (Connected Battery-Cables Ground Loose)))
    (HYP H4 (Connected Battery-Cables Solenoid Loose))
    (EXP H4 (Cause (Connected Battery-Cables Solenoid Loose)
  ((Not (Adjacent)) Starter-Gear Flywheel)))
    (TEST (Connected Battery-Cables Solenoid) (Type Visual)
  (Result (Connected Battery-Cables Solenoid Loose)))
    (HYP H5 (Power Battery Low))
    (TEST (Power Battery) (Type VAT40) (Result (Power Battery Low)))
    (HYP H6 ((Not (Generate)) Battery-Cell Power))
    (EXP H6 (Cause (Spill Battery-Cell Electrolytes)
  ((Not (Generate)) Battery-Cell Power)))
    (FAULT ((Not (Generate)) Battery-Cell Power))
    ))
```

Figure 4: An example protocol.

After learning, diagnosis is more efficient and more powerful for the same or similar problems, because the symptom-fault set can provide more reasonable hypotheses more quickly and the mental model is more capable of verifying an explanation.

2.4 Example

The general reasoning method is demonstrated by the following examples. The first of these examples shows learning with a symptom and hypothesis, without requiring an explanation. The symptom that the instructor reports is slow cranking of the engine. The system chains backward hypothesizing that the crankshaft is spinning slowly, that the starter motor is spinning slowly, that the battery isn't generating much electricity:

```
(crank engine slow) -->  ;
  (spin crankshaft slow) -->              ; cause
    (spin starter-gear slow) -->          ; cause
      (spin starter-motor slow) -->       ; cause
        (contains starter-wire current low) -->
          (generate battery electricity low)
```

The system achieves this backward chaining by using information about normal function and how, in general, modification of that normal function modifies the rest of the mechanism.

The instructor then suggests a hypothesis that a cracked wire can cause the observed symptom of slow cranking of the engine. The system chains forward, inferring that the battery cable may be carrying less than normal current, that the starter receives less power, and that the wire in the starter gets less electricity:

```
  (cracked battery-cable) -->
    (contains battery-cable current low) -->
      (input starter current low) -->
        (contains starter-wire current low)
```

In this case, the symptom and hypothesis chains meet, so an explanation is not needed. The system understands the instructor's hypothesis, and is able to learn a new symptom-fault set and add the relationship

```
(cause (cracked battery-cable) (crank engine slow))
```

to the mental model.

2.5 Example 2

As already mentioned, the system does not always have enough domain knowledge to fully understand an instructor's hypothesis. When an explanation is given, an important gap in the system's knowledge may be filled, thereby allowing completion of a causal chain. Additionally, the explanation may assist the system in directing the search for relevant causal relationships. For example, there are potentially many causal chains that might be constructed when the symptom is that the car is stalling. The explanation can help determine which of many chains is relevant.

Suppose that the instructor presents a situation in which the car has stalled. The system chains backward, hypothesizing as far as that there might not be combustion occurring in the cylinders:

```
(not (run engine)) -->      ; stalled engine
  (not (spin crankshaft)) -->
    (not (down-stroke cylinder)) -->
      (not (combustion cylinder))
; cylinder is a abstract term for any cylinder.
```

The instructor then provides the hypothesis that the butterfly valve of the choke assembly is stuck. Forward chaining would reveal:

```
  (not (movable butterfly-valve)) -->
    (flow air carburetor low)
```

In this case, the forward and backward chains do not meet, and the system does not know or cannot retrieve any causal relationships that might connect the chains. However, if the instructor provides the explanation that low air flow into the carburetor leads to a low air/gas mixture as the air passes the fuel float bowl then the following results:

```
  (not (movable butterfly-valve)) -->
    (flow air carburetor low)      -->
      (mix air gas less)           -->
        (not (combustion cylinder))    -->
          (not (down-stroke cylinder)) -->
            (not (spin crankshaft))    -->
              (not (run engine))
```

This example of explanation use demonstrates how a new causal relationship may be added, (flow air carburetor low) ⇒ (mix air gas less) , and how an existing causal relationship that was not accessed or was not known to apply, (mix air gas less) ⇒ (not (combustion cylinder)), can be used. The system, therefore, understands the instructor's hypothesis, and is able to learn a new symptom-fault set and add the relationships

```
(cause (flow air carburetor low)
       (mix air gas less))
(cause (not (movable butterfly-valve))
       (not (run engine)))
```

to the mental model.

2.6 Debugging

Since new information is being added to the mental model there is a possibility that a contradiction may occur. A con-

tradiction could arise in many different ways. There might be two pieces of knowledge such that

```
    (corroded battery-terminals) -->
            (connect battery battery-terminals)
and (corroded battery-terminals) -->
        (not (connect battery battery-terminals))
```

A circular chain might be possible from known information, such that

```
(corroded battery-terminals)
--> (not (connect battery battery-terminals))
--> (not (spin starter-motor))
--> (not (spin starter-gear))
--> (not (corroded battery-terminals))
```

There are actually many possibilities. For example,

```
(corroded battery-terminals) -->
    ((not (connect battery battery-terminals))
      & (spin starter-motor))
(not (connect battery battery-terminals)) -->
    (not (spin starter-gear))
(spin starter-motor) -->
    (spin starter-gear)
```

Contradictions may not be detected immediately because the causal information is distributed throughout the model, and because an arbitrary amount of causal chaining may be necessary. In cases where the new input is found to be inconsistent, the source of the information is used to decide what to believe. Knowledge from the expert is given precedence over older information. Information that contradicts the expert is either removed, as in implementation two below, or its strength is decreased, as in implementation one.

3 Implementations

Implementation of the general model described above has followed two parallel paths. This decision was made because the research is exploratory and, therefore should generate several alternative approaches to the problem. An additional advantage of the separate implementations is that the different programming efforts highlight different inconsistencies and difficulties with the model, possibly producing a more general theory than would arise from a single implementation.

3.1 First Implementation

One of the implementations is based upon a simple active semantic net, similar to local connectionist models such as McClelland and Rumelhart's [MR81] interactive activation model. In a diagnosis training example, this architecture receives input, consisting of symptoms, hypotheses, and explanations of th se hypotheses, as described above in the general algorithm section. The symptom will trigger backward chaining through the existing causal knowledge. This search is guided by weights on the causal links that indicate their credibility or likelihood. The underlying causal knowledge was designed with reference to Kuipers [Kui84] and DeKleer and Brown [dKB81], but is not as principled as were their approaches. To process the symptom, the system attempts to discover the cause.

As the instructor progresses in the problem, s/he will suggest possible hypotheses. For each, the system will try to explain why that hypothesis would cause the symptom, again the search is guided by weights on the causal links. If a causal chain can be found from the hypothesis to the symptom then a "short cut" cause is added, of the form *(cause hypothesis symptom)*. However, if the system is unable to find a causal chain, it attempts to bridge a gap between two partial chains. This is the situation depicted in section 2.4. When no chain is found, several possible incomplete paths may have been considered. If the end of an incomplete chain can be connected to the beginning of a chain from the symptom, then that causal gap will be filled and the entire chain collapsed as in section 2.4. Gaps will be filled only if there is an explanation or some general knowledge that indicates whether a cause is possible. An example of this second possibility is, a cracked wire can cause low electricity because (a) wires conduct electricity, and (b) a conduit affects what it conducts.

When explanations enter the system, they are assumed to indicate what causal gap filling is required. They may simply fill a gap themselves, or they may refer to causal knowledge that the system already has as indicated in section 2.5. The explanation is handled by continuing to process the previous hypothesis, emphasizing those chains that contain the explanation. Failing that, the system attempts to fill causal gaps from the end of all hypothesis chains to the provided explanation and from that explanation to the start of all backward chains from the symptom.

The system currently learns and partially debugs information necessary for diagnostic performance. It acquires new causal information, collapses chains of reasoning for more efficient use, and reduces the weights of causes that contradict what the instructor states or implies is a true causal relationship. In other words, the system learns what it does not know and specifically what it learns depends upon the inferences that can be made from the instructor's statements.

Future directions will include adding more principled causal representations, allowing more learning about the instructor's strategies from the explanations, and allowing learning of the abstract knowledge necessary to evaluate gap filling in the absence of an explanation.

3.2 Second Implementation

The second implementation uses an enhanced version of the mental model described in Allison [All87]. The process closely follows the general concept presented in section 2.2. A difference is that the system learns something even when the explanation does not enable bridging the gap between chains. For example, after steps 1 through 4 have been followed, the resulting chains could look like this:

```
H       AB  X   Y   Z    S
.------->.<--.--->  <----.
```

where the uppercase letters represent pieces of knowledge that are involved in causal chains:

```
H - Hypothesis
S - Symptom
X - Explanation
A - Edge of chain forward from hypothesis
B - Edge of chain back from explanation
Z - Edge of chain back from symptom
Y - Edge of chain forward from explanation
```

If either direction of chaining is successful in closing a gap, then a causal relationship can be learned. In the example, since *A* and *B* represent the same thing, the chains from *H to A* and from *E to B* meet, and the partial cause *(cause H E)* can be learned.

In the direction which cannot be closed, the system infers that the gap between the chains can be filled, based on the expert instructor having given the hypothesis and the symptom. In the example, nothing in the chain *E to Y* matches anything in the chain *S to Z*. The inference is made that smallest gap should be filled (*(cause Y Z)*).

The system learns different information when the mental model is in a different state. When the system is re-run using the same learning episode and the modified working model, additional information is learned. The system is able to build the chain further back from the symptom because of the newly learned causal knowledge, and similarly, is able to forward chain further from the hypothesis. This may allow the chains to meet, so the hypothesis can be associated with the symptom in the symptom-fault table, and the mental model can be updated to include that the hypothesis causes the symptom. This enables better diagnosis as well as more future learning.

The system resolves contradictions in the mental model by checking for a contradiction whenever causal information is added. The check is only done for the frame where the information is being added; no causal chaining is done. If a conflict exists, the piece of information with the higher credibility will be retained.

One thing shown by this implementation is that the program should have an understanding of the diagnostic strategy being used by the instructor in order to be better able to understand the explanation.

3.3 Summary of the Implementations

It is surprising that the two implementations, which began as vastly different representations of the same problem, have started to converge at many levels. The two implementations use different representations, one, a fine grained connectionist system and the other, more knowledge based. As the implementations proceeded, however, the fine grained approach has come to use the conventions of frame-based systems, and the knowledge-based approach has come to use a more homogeneous representation for all system knowledge. Similarly, through independent development, both approaches have come to use weights on causal relationships to represent the credibility of a cause. Of more practical importance for future research, however, are the differences. There are two general differences in the technical details of how the general algorithm was implemented. First, when a *causal gap* is present but no explanation fills that gap, implementation one uses generic knowledge about what affects what, whereas implementation two uses a less general but far simpler notion of filling gaps between recent hypotheses and symptoms. The first method brings more knowledge to bear on the postulation of a new causal relationship and should therefore lead to more reliable relationships and a more flexible metric for evaluating whether a given *causal gap* should be filled. The second difference involves where causal information is stored and how it can be accessed. Implementation one does not address the issue of limited availability of causal relationships. Instead, if a relationship is in the system, it can be used. On the other hand, implementation two allows the more realistic situation in which causes are not maximally indexed when they enter the system. That is, they may not necessarily be retrieved when needed unless the proper cues are present. This is a more realistic and efficient approach for a system with a very large memory.

A major direction of future research will involve integrating the two implementations into a single general method.

4 Related Work

As has already been noted, the current effort uses techniques that are in some ways similar to those of Dejong and Mooney [DM86] and Mitchell et al. [MKK86]. It is similar in that it recognises that a system may know all the necessary information to solve a given problem, but a solution path through this information may be inefficient to calculate. Instruction or observation might allow a sample path to be generated and condensed to a more easily used form. Specifically, in diagnosis, a causal chain must be discovered in a potentially very large network of causal information. EBL can be profitably used to permit instruction to produce "short cuts" in that network.

Classical EBL, however, does not produce enough learning when the causal network is incomplete. This may be remedied by the learning by failing to explain (LBFE) [Hil86] technique of isolating the information that is present in the input but is not understood, and subsequently adding it to the existing EBL system. Although the current model has not yet been described in exactly these terms, it is in fact an example of LBFE. It differs from Hill's work by proposing that the information that must be added in the absence of an explanation is not necessarily explicitly represented in the input. Instead, the current effort allows for more general assignment of blame by disambiguating what might be implied by the instructor. Also, the current effort presents a domain independent notion of LBFE that describes how potentially any causal net might grow, whereas Hill's effort was, in his own view, domain specific.

One of the methods that is used to augment incomplete networks in the current approach is to use relationships that are more general than causes in order to infer causation. For example, an action and a state change that relate to the same object tend to be causally related. This technique was originally used by Pazzani [Paz87] and a similar approach was suggested by Russell [Rus87].

Other studies that have shared some of the goals of the current research in other domains include, Haas and Hendrix ([HH83], Learning by being told), and Sammut and Banerji ([SB86], Concept learning by asking questions).

5 Conclusions and Future Directions

This paper has discussed the Learning by Understanding Explanations paradigm that was observed by Lancaster and Kolodner [LK87] in the training of car mechanics. The main contribution of the current effort is in the ability to accept new knowledge and make it part of the mental model, while using it to understand an explanation and form a new generalization.

There are several directions for future research. First, in diagnosis, causal chaining is not the only strategy used, though it was the most common in the protocols. The explanations used by the instructor reflect several different strate-

gies. For example, a dead battery could be hypothesized as a problem because that occurs frequently. It may be that a particular model of car has a defect. Or the explanation may be an explanation of a normal function of a component that the instructor thinks the student may not understand. For this reason, and to allow strategies to be learned and improved, diagnostic strategies must be explicitly represented. Some initial work has been done on the representation.

Second, better representation of the causal knowledge is needed to take full advantage of the inferencing possible from qualitative models. The aim is to use more levels of abstraction to allow reasoning at whatever level may be appropriate, and so that knowledge of the normal function of a mechanism can be more useful in inferencing. Third, more learning may be possible in this paradigm if Case-Based Reasoning [KJ84], a method of using previous episodes and evaluation of their results to suggest solutions to new problems, could be integrated. Lastly, Larry Barsalou and Chris Hale of the Psychology department at the Georgia Institute of Technology are planning experiments to further investigate how people do diagnosis of mechanical devices. Future versions of the current model will attempt to incorporate those findings.

References

K. R. Allison. Use of a working model in fault diagnosis. In *Proceedings of the 25th Annual Conference of the Southeast Region ACM.*, 1987.

G. DeJong. Acquiring schemata through understanding and generalized plans. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence.*, Karlsruhe, West Germany., 1983.

J. de Kleer and J. S. Brown. Mental models of physical mechanisms and their acquisition. In J. R. Anderson, editor, *Cognitive Skills and Their Acquisition.*, Lawrence Erlbaum, Hillsdale, NJ, 1981.

G. DeJong and R. Mooney. Explanation based learning: an alternative view. *Machine Learning*, 1:145–176, 1986.

R. E. Fikes, P. Hart, and N. J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972.

N. Haas and G. Hendrix. Learning by being told: acquiring knowledge for information management. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, Morgan Kaufmann, Los Altos, CA, 1983.

R. Hill. Learning by failing to explain. In *Proceedings of the National Conference on Artificial Intelligence.*, 1986.

J. Kolodner and R. Simpson Jr. A case for case-based reasoning. In *Proceedings of the Sixth Annual Conference of the Cognitive Science Society*, 1984.

J. Kuipers. Commonsense reasoning about causality: deriving behavior from structure. *Artificial Intelligence*, 24:169–203, 1984.

J. Lancaster and J. Kolodner. Problem solving in a natural task as a function of experience. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, Irvine, CA, 1987.

M. Minsky. A framework for representing knowledge. In P. H. Winston, editor, *The Psychology of Computer Vision*, McGraw-Hill, New York, 1975.

T. M. Mitchell, R. M. Keller, and S. T. Kedar-Cabelli. Explanation based learning: an unifying view. *Machine Learning*, 1:47–80, 1986.

J. L. McClelland and D. E. Rumelhart. An interactive activation model of context effects in letter perception: part 1. an account of basic findings. *Psychological Review*, 88:375–407, 1981.

M. Pazzani. Inducing causal and social theories: a prerequisite for explanation-based learning. In *Proceedings of the Fourth Annual International Workshop on Machine Learning*, Irvine, CA, 1987.

S. J. Russell. Analogy and single-instance generalization. In *Proceedings of the Fourth Annual International Workshop on Machine Learning*, Irvine, CA, 1987.

C. Sammut and R. B. Banerji. Learning concepts by asking questions. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach, Volume II*, Morgan Kaufmann, Los Altos, CA, 1986.

The Use of Explanations for Completing and Correcting Causal Models[1]

From the Proceedings of the 10th annual Cog. Sci. Conf., 1988

Joel D. Martin and Michael Redmond
Georgia Institute of Technology
E-mail: joel@gatech.edu, redmond@gatech.edu

Abstract

Causal models describe some part of the world to allow an information system to perform complex tasks such as diagnosis. However, as many researchers have discovered, such models are rarely complete or consistent. As well, the world may change slightly, making a previously complete model incomplete. A computational theory of the use of causal models must allow for completion and correction in the face of new evidence. This paper discusses these issues with respect to the evolution of a causal model in a diagnosis task. The reasoner's goal is to diagnose a fault in a malfunctioning automobile, and it improves its diagnostic model by comparing it with an instructor's. A general process model is presented with two implementations. Related work in explanation based learning and in incorrect causal models is discussed.

Keywords: Learning, Causal Models, Explanations, Diagnosis

INTRODUCTION

A causal model or domain theory is an essential ingredient in understanding complex situations. For example, in order to diagnose a fault in a complex system, the diagnostician must be capable of making guesses about what might be wrong. However, without appropriate heuristic knowledge to guide and make those guesses, the correct hypothesis may never arise. Although many researchers have acknowledged the need for such causal models [Kuipers, 1984] [deKleer & Brown, 1981], very few have been concerned with the possibility that the domain theory may be incomplete or inconsistent. Those researchers who have recognized this problem [Rajamoney & DeJong, 1987] have not yet allowed for modification of the underlying causal theory.

With this in mind, Lancaster and Kolodner [1987] took protocols of the diagnostic behavior of novice, intermediate, advanced, and expert car mechanics. They observed evidence for a working model, a set of symptom fault pairings, and diagnostic strategies. They also observed [Lancaster, personal communication] that less experienced mechanics had inconsistent and incomplete knowledge, as one might expect. The research presented in this paper represents an effort to discover how an incomplete causal model (novice) can evolve to a more complete (experienced) state as a result of problem solving experience coupled with explanations about how those problems are solved.

Our model is implemented in two computer programs called EDSEL-1 and EDSEL-2 (Explanation in Diagnosis: the use of Symptoms, hypotheses, and Explanations for Learning) that each begin with a novice memory and are presented with problems and explanations of how to solve those problems. Specifically, the systems "watch" or attend to an instructor who is diagnosing a fault in an automobile. As they do so, they attempt

[1]This research was supported by the Army Research Institute for the Behavioral and Social Sciences under Contract No. MDA-903-86-C-173. The authors wish to thank Janet Kolodner for her advice and guidance, and Mark Graves and Hong Shinn for helpful comments on earlier versions of the paper.

to identify missing information of various types or to identify whether there is an inconsistency. If one of these problems is discovered, the systems modify their causal model to prevent the difficulty in the future. The recognition and modifications are based upon an attempt by the systems to explain their input.

The paper describes a general algorithm, presents the issues involved in completing and correcting causal models, and compares the two implementations.

GENERAL PROCESS

Completing and correcting a causal model requires a reasoner to recognize when it is missing a piece of information and then to incorporate that information into the model. This notion is complicated by the fact that there are different types of information in the model and that existing knowledge affects how new information is incorporated.

Redmond and Martin [1988] noted in the protocols from Lancaster and Kolodner [1987] that an instructor provides the students with a symptom, a series of hypotheses, and explanations for those hypotheses. We demonstrated that a system may process these inputs by attempting to build causal chains between hypotheses and the symptom, using provided explanations if no causal chain is obvious. If a complete chain can be built, it can be collapsed and be used more efficiently in future similar situations. If a complete chain cannot be built, then the instructor's explanation can be helpful, either by being added directly to the causal model, or by allowing the gap in the chain to be bridged. Our name for this process is Learning by Understanding Explanations (LBUE). The causal model contains frames [Minsky, 1975] for the components of a car, with slots for inputs, outputs, connections, parts, functions, and causal relationships between structures. For example, one piece of the current model is:

```
Starter:
   (isa component)                              ;A starter is a component.
   (part-of starting-system)                    ;Is a part of the starting system.
   (input electricity battery battery-cables)   ;Electricity from battery via cables.
   (parts starter-pinnion-gear starter-motor)   ;PARTS: pinnion gear and starter motor.
   (function spin-action starter-pinnion-gear) ;FUNCTION: spin the pinnion gear.
   (cause (switch-action solenoid on)           ;Solenoid switch causes the two gears to interlock.
       (interlock starter-pinion-gear flywheel-ring-gear))
   (cause (crank starter-pinion-gear)           ;Cranking one gear causes the other to crank.
       (crank flywheel-ring-gear))
```

The causal chaining process uses the causal relationships and some of the related knowledge. Besides the causal model of the domain, the LBUE approach also includes a set of likely symptom-fault pairings, as observed by Lancaster and Kolodner [1987] . These sets of pairings associate a symptom with a problem, and are used to derive initial hypotheses during diagnosis, and to index into the causal model at the appropriate place. The general algorithm for the process is as follows:

1. From the symptom (presented by the instructor), chain backward, inferring possible findings that could lead to the symptom.
2. From each hypothesis (presented by the instructor), chain forward, inferring possible effects that could be caused by the hypothesized fault.
3. If the symptom chain meets a hypothesis chain, then the reasoner has an explanation for the hypothesis, and the generalisation that (CAUSE HYPOTHESIS SYMPTOM) is added to the symptom fault table and to the causal model.

4. If the chains do not meet - the reasoner does not have enough information to explain the hypothesis. In this case, it uses an explanation presented by the instructor,
   (a) Chain backwards from the explanation toward the hypotheses chain.
   (b) Chain forward from the explanation toward the symptom chain.
   (c) If both directions can be linked, then the most general relationship (CAUSE HYPOTHESIS SYMPTOM) can be learned.
5. Add explanation to the causal model.

The LBUE process results in an updated causal model. As discussed in following section, the things that may be learned are,

1. new objects
2. new relationships between objects
3. new causal information
4. new symptom fault knowledge

After learning, diagnosis is more efficient and more powerful for the same or similar problems, because the symptom-fault set can provide more reasonable hypotheses more quickly and the causal model is more capable of verifying an explanation. In addition, what the reasoner learns depends on what it already knows, since the reasoner's ability to chain back from the symptom and forward from the hypothesis is affected by the knowledge in the causal model. This means that the chains could meet given one version of the causal model, and have an unbridgable gap given another version.

The alternative to the LBUE approach is simply to remember symptom-hypothesis pairs. However, this would require a system to have already experienced a fault in order to diagnose it; no general knowledge is retained.

ISSUES FOR COMPLETING AND CORRECTING CAUSAL MODELS

As outlined above, a good diagnostic reasoner tries to explain why an hypothesis causes a symptom. It is this process that allows for the recognition of different types of missing information, and mediates the addition of knowledge to the causal model. The process of explaining hypotheses identifies missing information that might be useful for diagnosis because diagnosis is itself explanation, and hence requires the same information.

TYPES OF MISSING KNOWLEDGE

In general, a causal model may be missing many causal relations necessary for diagnosis. A reasoner will recognize that a causal relationship is missing if an explanation of a symptom cannot be formed, either while watching an instructor or while doing diagnosis. As well, there are situations in which an unknown causal relationship will be presented to the reasoner. Both possibilities are simple to detect, the former when causal chaining fails or no reasonable hypothesis is generated, and the latter, when the reasoner is actually told that something is missing.

Another form of knowledge whose absence is easily detected consists of referred-to facts. In other words, when an object or general relationship between objects is asserted, but is not known, then it is missing from the model. Somewhat more interesting are implied facts. The reasoner guesses it is missing an implied fact when a causal relationship is stated or implied by an instructor that the reasoner believes requires a mediating fact. For example, a reasoner may know,

```
(INTERLOCKED gear1 gear2) & (SPIN gear1 'clockwise) --> (SPIN gear2 'c-clockwise)
```

and an instructor may state,

```
(SPIN starter-gear 'clockwise) --> (SPIN flywheel-ring-gear 'c-clockwise)
```

From this, the reasoner will recognize that it is missing a fact (i.e., that the two gears are interlocked).

The final type of information that a reasoner may be missing is essentially efficiency information. The reasoner must be able to arrive at a reasonable or correct hypothesis quickly. If it cannot, the causal model must be modified to ensure timely and correct diagnoses in the future. The reasoner can recognize that it is missing this kind of information if it arrives at an incorrect hypothesis during diagnosis or if its hypotheses differ from the instructor's.

METHODS OF HANDLING INCOMPLETE KNOWLEDGE

An instructor's explanation of a given hypothesis can lead to information being added in three different ways. The explanation itself could be an unknown causal relationship which can be added to the model directly. For example, if the instructor explained

```
(cause (corroded battery-terminals) (not (connect battery battery-terminals)))
```

and this relationship was not associated with either battery or battery-terminals in the causal model, then it can be added there. A second way that the instructor's explanation can be used is to enable filling a gap in a causal chain. Either the explanation filled the gap, or it was a better cue to information that was not being accessed in the causal model. If the causal chain that can be built from the symptom (NOT (RUN ENGINE)) is:

```
(not (run engine)) -->  (not (spin crankshaft)) -->
    (not (down-stroke cylinder)) --> (not (combustion cylinder))
```

and the causal chain that can be built from the associated hypothesis (NOT (MOVABLE BUTTERFLY-VALVE)) is:

```
(not (movable butterfly-valve)) --> (flow air carburetor low)
```

then there is a gap in the causal chain — the hypothesis is not fully explained. If the instructor provides the explanation that low air flow into the carburetor leads to a low air/gas mixture as the air passes the fuel float bowl then the following results:

```
(not (movable butterfly-valve)) --> (flow air carburetor low) -->
  (mix air gas less)  -->  (not (combustion cylinder)) -->
     (not (down-stroke cylinder)) --> (not (spin crankshaft)) --> (not (run engine))
```

Not only is the causal relationship given in the explanation used in filling the gap, but the relationship that (MIX AIR GAS LESS) causes (NOT (COMBUSTION CYLINDER)) is accessible when it hadn't previously been accessible, since the cue of carburetor is now available. Between the two, the gap has been filled.

The third way in which the instructor's explanation can be used is to infer a relationship that would fill a gap in a causal chain. If the explanation doesn't allow bridging the gap as discussed above, causal relationships which bridge the gap, which are implied by the expert instructor, can be inferred. The instructor implies that there is a causal relationship between the hypothesis and symptom and that the explanation lies along this causal chain.

Gaps will be filled with inferences if there is some general knowledge that indicates a cause is possible. For example, a cracked wire can cause low electricity because (a) wires conduct electricity, and (b) a conduit affects what it conducts. This would be given lower credibility than other learned relationships. There may be several plausible but inconsistent inferences that might fill a gap; the one chosen could depend on confirmation from a human observer.

Knowledge can be added to an incomplete causal model by inferring facts from a cause. This would occur as a result of the starter-gear ring-gear example mentioned above. In this case, the reasoner will infer that the starter gear and ring gear are interlocked.

In a sense, inefficiently represented knowledge is a type of incomplete knowledge. The information that is needed is in the causal model, but is not useful because it cannot be accessed, or it is given insufficient credibility, or it leads to slow processing. For instance, the explanation can allow the access of knowledge that couldn't previously be accessed. Additionally, filling a gap in a causal chain, as discussed above, is a way of dealing with some inefficient knowledge. This allows collapsing the chain into a single causal relationship, which can be used for more efficient processing. In collapsing the chains, the LBUE method has some similarities to Explanation-Based Learning (EBL) [Mitchell, Kellar, & Kedar-Cabelli, 1986] [DeJong & Mooney, 1986]. In order to allow for proper generalization of variables [DeJong & Mooney, 1986], a substitution list is kept that indicates to what categories each feature in the example was matched in order to instantiate the causal relationships. The collapsed chain then uses the most general category for a feature as the variable name in the antecedent or consequent of the new causal relationship.

INCONSISTENT KNOWLEDGE

Since new information is being added to the causal model, there is a possibility that a contradiction may occur. A few types of contradiction are possible. In one case, the same condition could be believed to cause contradictory effects such as:

```
(corroded battery-terminals) --> (connect battery battery-terminals)
& (corroded battery-terminals) --> (not (connect battery battery-terminals))
```

Alternatively, a chain might be possible from known information, such that a condition indirectly causes a contradiction of the condition.

Contradictions may not be detected immediately, though, because the causal information is distributed throughout the causal model, and because an arbitrary amount of causal chaining may be necessary to detect the contradiction.

In cases where the new input is found to be inconsistent, the source of the information can be used to decide what to believe. Knowledge from the expert is given precedence over older information. Information that contradicts the expert is either removed or its strength is decreased, depending upon implementation.

IMPLEMENTATIONS

Implementation of the general model described above has followed two parallel paths. This decision was made because the research is exploratory, and therefore should generate several alternative approaches to the problem, and highlight different inconsistencies and difficulties with the model.

EDSEL-1 is based upon a simple active semantic net, similar to local connectionist models such as McClelland and Rumelhart's [1981] interactive activation model. EDSEL-

2 uses an enhanced version of the causal model described in Allison [1987] . Although the process in both implementations closely follows the general model presented above, there are two significant differences. First, when a *causal gap* is present but no explanation fills that gap, EDSEL-1 uses generic knowledge about what affects what, whereas EDSEL-2 uses a less general but far simpler notion of filling gaps between recently proposed forward chains from hypotheses and backward chains from the symptom. The first method is a more flexible metric for evaluating whether a given *causal gap* should be filled, and therefore should lead to more reliable causal relationships. The second difference involves where causal information is stored and how it can be accessed. EDSEL-1 does not address the issue of limited availability of causal relationships, whereas EDSEL-2 allows the more realistic situation in which causes are not maximally indexed when they enter the system. That is, they may not necessarily be retrieved when needed unless the proper cues are present. This is a more realistic and efficient approach for a system with a very large memory.

RELATED WORK

Rajamoney and DeJong [1987] has specifically addressed the problem of inconsistencies or missing information in a causal model for simulation. If more than one simulation is possible, his system will experimentally search for disambiguating features in the envirc ıment. Although this approach is clearly useful, it does not allow for modification of the general causal information in the model. It concentrates on quantitative values for the current situation, and does not learn any general knowledge.

As has already been noted, in order to update causal models, the current effc rt uses an explanation based technique that is in some ways similar to those of DeJong and Mooney [1986] and Mitchell et al. [1986] . Specifically, in diagnosis, a causal chain must be discovered in a potentially very large network of causal information. EBL can be profitably used to permit instruction to produce "short cuts" in that network.

Classical EBL, however, does not produce enough learning when the causal network is incomplete. This may be remedied by the learning by failing to explain (LBFE) [Hall, 1986] technique of isolating the information that is present in the input but is not understood, and subsequently adding it to the existing EBL system. Although the current model has not yet been described in exactly these terms, it is in fact an example of LBFE. It differs from Hall's work by proposing that the information that must be added in the absence of an explanation is not necessarily explicitly represented in the input. Also, the current effort presents a domain independent notion of LBFE that describes how potentially any diagnostic causal net might grow, whereas Hall's effort was, in his own view, domain specific.

One of the methods that is used to augment incomplete networks in the current approach is to use relationships that are more general than causes in order to infer causation. For example, an action and a state change that relate to the same object tend to be causally related. This technique was originally used by Pazzani [1987] and a similar approach was suggested by Russell [1987] .

CONCLUSIONS AND FUTURE DIRECTIONS

This paper has discussed the LBUE paradigm for dealing with an incomplete or inconsistent causal model in the training of car mechanics. The main contribution of the current effort is in the ability to accept new knowledge and incorporate it into the causal model, while using it to understand an explanation and form a new generalization.

There are several directions for future research. First, in diagnosis, causal chaining is not the only strategy used, though it was the most common in the protocols. The explanations used by the instructor reflect several different strategies. For this reason, and to allow strategies to be learned and improved, diagnostic strategies must be explicitly represented. Some initial work has been done on this representation.

Second, better representation of the causal knowledge is needed to take full advantage of the inferencing possible from qualitative models. The aim is to use more levels of abstraction to allow reasoning at whatever level may be appropriate. Third, more learning may be possible in this paradigm if Case-Based Reasoning [Kolodner & Simpson, 1984], a method of using previous episodes and evaluation of their results to suggest solutions to new problems, could be integrated.

REFERENCES

References

Allison, K. R. (1987). Use of a working model in fault diagnosis. In *Proceedings of the 25th Annual Conference of the Southeast Region ACM.*

DeJong, G. & Mooney, R. (1986). Explanation based learning: an alternative view. *Machine Learning, 1,* 145–176.

de Kleer, J. & Brown, J. S. (1981). Mental models of physical mechanisms and their acquisition. In J. R. Anderson (Ed.), *Cognitive Skills and Their Acquisition.* Hillsdale, NJ: Lawrence Erlbaum.

Hall, R. (1986). Learning by failing to explain. In *Proceedings of the National Conference on Artificial Intelligence.*

Kolodner, J. & Simpson, R. (1984). A case for case-based reasoning. In *Proceedings of the Sixth Annual Conference of the Cognitive Science Society.*

Kuipers, J. (1984). Commonsense reasoning about causality: deriving behavior from structure. *Artificial Intelligence, 24,* 169–203.

Lancaster, J. & Kolodner, J. (1987). Problem solving in a natural task as a function of experience. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society.*

Lancaster, J. (personal communication). August, 1987.

McClelland, J. L. & Rumelhart, D. E. (1981). An interactive activation model of context effects in letter perception: part 1. an account of basic findings. *Psychological Review, 88,* 375–407.

Minsky, M. (1975). A framework for representing knowledge. In P. H. Winston (Ed.), *The Psychology of Computer Vision.* New York: McGraw-Hill.

Mitchell, T. M., Keller, R. M., & Kedar-Cabelli, S. T. (1986). Explanation based learning: an unifying view. *Machine Learning, 1,* 47–80.

Passani, M. (1987). Inducing causal and social theories: a prerequisite for explanation-based learning. In *Proceedings of the Fourth Annual International Workshop on Machine Learning.*

Rajamoney, S. A. & DeJong, G. F. (1987). *Active ambiguity reduction: An experimental design approach to tractable qualitative reasoning.* Technical Report UILU-ENG-87-2225, University of Illinois at Urbana-Champaign.

Redmond, M. & Martin, J. (1988). Learning by understanding explanations. In *Proceedings of the 26th Annual Conference of the Southeast Region ACM.*

Russell, S. J. (1987). Analogy and single-instance generalization. In *Proceedings of the Fourth Annual International Workshop on Machine Learning.*

In ACM Special Interest Group on Artificial Intelligence (SIGA
Special Issue on knowledge Acquisition

Acquiring Knowledge by Explaining Observed Problem Solving[1]

Joel D. Martin and Michael Redmond
Georgia Institute of Technology
E-mail: joel@gatech.edu, redmond@gatech.edu

Keywords: apprenticeship, instruction, explanations, model-based, automatic

Abstract

Learning by Understanding Explanations (LBUE) can be an important tool for automatic knowledge acquisition in diagnostic domains. It takes fuller advantage of an expert's knowledge than some more automatic approaches and uses model-based reasoning to reduce brittleness. LBUE specifically integrates the learning of efficiency information as in Explanation Based Generalization (EBG) with further learning of the causal model. Expert protocols are coded into a machine readable form and then input to the LBUE process. The result of learning is that the system becomes capable of solving the same problem in the future and is also able to apply any newly learned causal model information to novel but similar cases.

1 Introduction

Experts use many rules to easily wade through the complexities of their domain. However, capturing these rules for expert systems is a difficult, time consuming task. Traditionally, the rules have been handcoded from problem solving protocols or through some other interactions with the expert. A natural inclination of expert systems practitioners when faced with this task is to try to totally automate the knowledge acquisition process. For example, several approaches have considered the possibility of having an intelligent learning system, like ID3 [Quinlan, 1983], discover the appropriate rules by examining solved examples. These approaches, however, ignore the value of the expert as a guide for learning. Having a Londoner guide one through the streets of London is far superior to an extensive search through all possible routes from one point to another. A more appropriate automatic solution to knowledge acquisition should take advantage of an expert's experience. This paper presents such a system, EDSEL (Explanations and Diagnosis: Symptoms, hypotheses, and Explanations for Learning), that acquires diagnostic knowledge through a process termed Learning by Understanding Explanations (LBUE) [Redmond & Martin, 1988; Martin & Redmond, 1988].

The obvious analogy of this type of knowledge acquisition with human learning is 'apprenticeship', where a novice becomes more expert by learning from watching an instructor. Learning apprentice systems [Mitchell, 1985; Wilkins, 1988], in general, allow incremental learning from examples, but also allow the instructor to guide what is learned. This means that the system learns more than simple associations between problem and solution. ODYSSEUS [Wilkins, 1988], for example, learns something about what questions to ask. This extra learning is achieved by using an underlying model to explain some behavior of the experts and to use that explanation as a hint for learning. Some systems use the explanation to make the model information more efficient, while others actually allow improvements of the model. Both approaches are clearly necessary: increased efficiency leads to improved performance and a flexible model permits continuous learning and less brittleness. No systems besides the one described below have yet demonstrated both of these types of learning.

The apprenticeship system EDSEL receives input of three types: symptoms, an expert's hypotheses, and partial expert explanations. Using a background causal model, it attempts to explain how a given hypothesis can literally cause the symptom. If the model is sufficiently complete to allow the explanation, then that symptom and hypothesis can be more efficiently paired to improve later performance. If, on the other hand, the

[1]This research was supported by the Army Research Institute for the Behavioral and Social Sciences under Contract No. MDA-903-86-C-173. The authors wish to thank Janet Kolodner for her advice and guidance, and Stephen Robinson and anonymous reviewers for helpful comments on earlier versions of the paper.

model is incomplete, the system uses partial explanations and known constraints to augment the model and allow full explanation of the given hypothesis. EDSEL's algorithm allows it to use expert guidance to increase efficiency, to permit continuous learning, and to reduce system brittleness.

2 Protocols

While this approach shares the idea of using verbal protocols of an expert's problem solving with many knowledge acquisition efforts [Johnson 1983], the *use* of the protocols varies substantially from other approaches. EDSEL, and the learning method LBUE which it embodies, were inspired by the protocol analysis work of Lancaster and Kolodner [1987, 1988]. They took verbal protocols of car-mechanics students trying to solve problems, and of the instructor solving the same problems afterwards for the students' benefit. This is the normal mode of teaching for the school, after background classroom instruction. It is a close relative to apprenticeship learning. The statements in the protocols were each coded as one of, hypotheses, rules, information gathering, observations, and restatements. From these, Lancaster and Kolodner identified five types of learning observed in the students, *Learning by understanging explanations, Active gap filling, Learning from interpreting feedback, Abstraction,* and *Case-Based Reasoning.* We have focused, to this point, on learning by understanding explanations. To permit LBUE to learn from the instructor without natural language processing capability, we transform the instructor's protocols into machine analyzable form, specifying symptoms, hypotheses, explanations of hypotheses, tests, and faults in a predicate notation. This serves as input to the LBUE process, to allow knowledge to be added to the background causal model, and to permit collapsing of that knowledge into rules for greater efficiency. The system avoids the need to manually extract rules that were implicit, and the need to manually infer steps that were skipped in the instructor's report of his/her problem solving.

The process of coding the instructor's protocols into machine analyzable form is a manual process, but not a difficult one. The most important thing is to establish a vocabulary of predicates for the domain, which can be used for expressing normal function and malfunctions. For example, in the auto-mechanic domain some of the predicates included: adjacent, amount, available, bent, broken, charge, clean, click, clogged, closed, compression, conduct, connected, contains, corroded, covered, cracked, crank, crossed, generated, hesitate, pressure, resistance, and tension. It is important to get coverage without redundancy, so that the relations can be expressed and that similar things are expressed similarly. The predicate vocabulary can then be used in coding the protocols and the initial causal model. Given the vocabulary, coding the protocols is only a matter of identifying the symptoms, hypotheses, and explanations of hypotheses, and expressing them in terms of the predicates and components. As will be seen, the components do not even necessarily have to be included in the model.

We will show one example from our domain to demonstrate this. One protocol contained the following:

> "Drivability complaint is that the engine turns over very slow while cranking. ... See, these connections to the battery could be loose, could be all corroded, right? That's common with battery terminals. You might have a voltage drop between post and clamp. We may have a loose cable on the starter ... So, the problem that we have is either in the charging system or starting ..."

This was put into a variable for input to EDSEL as:

```
(setq *protocol#13*
 '((SYM (Slow (Crank Engine-system)))
  (HYP H1 (Not (Connected Battery Battery-Cables)))
  (HYP H2 (Corroded Battery-Terminals))
  (EXP H2 (High (Frequency
                 (Corroded Battery-Terminals))))
  (EXP H2 (Cause
            (Corroded Battery-Terminals)
            (Low (Contains Battery-Cables Voltage))))
  (HYP H3 (Loose (Connected Battery-Cables Starter)))
  (HYP H4 (Malfunction Charging-System))
  (HYP H5 (Malfunction Starting-System))
  ...
 ))
```

The success of this approach is especially encouraging because the protocols used were not taken with this approach in mind. In future efforts to use this technique, we would want to emphasize to the instructor that s/he especially report all hypotheses, along with a brief explanation of those hypotheses.

3 Causal Model

The causal model represents components of the car in a manner roughly equivalent to frames [Minsky 1975] or objects [Goldstein 1980]. Information about a component includes what it is part of, what its parts are, its inputs and outputs, and the causal relations between it and other components. For example, Figure 1 shows a representation of a starter. An important advantage of using the model is that besides allowing the learning of rules, the problem solver can fall back on the model when there are no available rules for a particular problem, thus making a more robust, less brittle problem solver. Chandrasekaran and Mittal's [1982] MDX was similar in that it generated rules from a model of normal function, which made it less brittle. However, it did not continue to improve its model.

The system is given an initial model roughly equivalent to the classroom instruction given the technical school students. This information can come from an in-

tructory text, and is mainly information about nor-

```
Starter:
   (isa component)
        ;A starter is a component.
   (part-of starting-system)
        ;Part of the starting system.
   (input electricity battery battery-cables)
        ;A starter receives electricity from
        ;the battery via battery cables.
   (parts starter-pinion-gear starter-motor)
        ;A starter has parts: pinion gear and
        ;starter motor.
   (function spin-action starter-pinion-gear)
        ;The function of the starter is to
        ;spin the pinion gear.
   (cause (switch-action solenoid on)
          (adjacent starter-pinion-gear
                    flywheel-ring-gear))
        ;Setting the solenoid switch causes
        ;the pinion gear and the flywheel gear
        ;to become adjacent.
   (cause (crank starter-pinion-gear)
          (crank flywheel-ring-gear))
        ;Cranking the pinion gear causes the
        ;flywheel gear to crank.
```

Figure 1: A Generalized Mental Model Definition of a Starter.

mal function. It is expressed using the vocabulary of predicates established for the domain. It's not necessary for this information to be complete, the system can learn even with a minimal amount of knowledge. Therefore this approach does not assume that the 'student' retains all the knowledge presented in the classroom. It does, however, follow the intuition that the more knowledge the 'student' has, the better the learning will be. A student with more knowledge can learn more specific new knowledge from the experts' examples.

In the initial model, several types of knowledge can be missing. These include:

- Causal relationships
- Other relationships between objects
- Objects.

This also includes poor organization of the knowledge that would lead to inefficiency in diagnosis. The LBUE process allows learning of each of these types of knowledge, as well as allowing retraction of incorrect pieces of knowledge.

3.1 Causal Relationships

The most important type of missing knowledge is 'causal relations'. In general, a causal model may be missing many causal relations that are necessary for diagnosis. A reasoner will recognize that a causal relationship is missing if an explanation of a symptom cannot be formed, either while watching an instructor or while actually doing diagnosis. As well, there are situations in which an unknown causal relationship will be presented to the reasoner. Both possibilities are simple to detect, the former when causal chaining fails or when no reasonable hypothesis is generated, and the latter, when the reasoner is actually told that something is missing.

For instance, suppose that the instructor presents a situation in which the car has stalled. The reasoner attempts to determine what could lead to the symptom. This resembles a backward chain. In a particular circumstance, the reasoner could be missing causal relations and hypothesize only as far as that there might not be combustion occurring in the cylinders:

```
(not (run engine)) -->  ;stalled engine
   (not (spin crankshaft)) -->
      (not (down-stroke cylinder)) -->
         (not (combustion cylinder))
```

The instructor then provides the hypothesis that the butterfly valve of the choke assembly is stuck. The reasoner works forward trying to determine what that would cause. This resembles a forward chain. Because of causal knowledge missing from the reasoner's model, it only gets as far as a lack of air flow into the carburetor:

```
(not (movable butterfly-valve)) -->
   (low (flow air carburetor))
```

In this case, the forward and backward chains do not meet, and the reasoner does not know or cannot retrieve any causal relationships that might connect the chains. No complete explanation has been found for the hypothesis. However, if the instructor provides the explanation that low air flow into the carburetor leads to a low air/gas mixture as the air passes the fuel float bowl then the following complete explanation could result:

```
(not (movable butterfly-valve)) -->
  (low (flow air carburetor))   -->
    (low (mix air gas))         -->
      (not (combustion cylinder))     -->
        (not (down-stroke cylinder)) -->
          (not (spin crankshaft))     -->
            (not (run engine))
```

This example demonstrates how a new causal relationship may be added, (low (flow air carburetor)) ⇒ (low (mix air gas)), and how an existing causal relationship that was not accessed or was not known to apply, (low (mix air gas)) ⇒ (not (combustion cylinder)), can be used. The system, therefore, understands the instructor's hypothesis, and is able to learn a new symptom-fault association and add the relationships

```
(cause (low (flow air carburetor))
       (low (mix air gas)))
(cause (not (movable butterfly-valve))
       (not (run engine)))
```

to the mental model.

In the other case, when the explanation,

```
(cause (cracked distributor-cap)
       (contains distributor-cap moisture)),
```

contains knowledge that the mental model does not contain, the system only needs to check the model to realize that the information is missing.

3.2 Other Relationships

Another form of knowledge whose absence is easily detected consists of referred-to facts. These facts can involve various non-causal relationships. Specifically, when an object or general relationship between objects is asserted, but is not known, then it is missing from the model. Somewhat more interesting, though, are when such relationships are merely implied. The reasoner guesses it is missing an implied fact when a causal relationship is stated or implied by an instructor that the reasoner believes requires a mediating fact. For example, a reasoner may know,

```
(INTERLOCKED gear1 gear2) &
   (SPIN gear1 'clockwise) -->
      (SPIN gear2 'c-clockwise)
```

and an instructor may state,

```
(SPIN starter-gear 'clockwise) -->
  (SPIN flywheel-ring-gear 'c-clockwise)
```

From this, the reasoner will recognize that it is missing a fact (i.e., that the two gears are interlocked).

3.3 Objects

When the model does not know of the existence of an object, such as a component to the car, it becomes obvious when a hypothesis, symptom or explanation refers to the object and the system can not retrieve any information about the object when it tries to form an explanation. As long as the system does not need to reason about the object it can remain blissfully ignorant of its existence, thus avoiding the need to understand atoms and molecules.

3.4 Inefficiency

In a sense, inefficiently represented knowledge is a type of incomplete knowledge. The information that is needed is in the causal model, but is not useful because it cannot be accessed, or it is given insufficient credibility, or it leads to slow processing. In this situation, the model is said to be missing efficiency information. The reasoner must be able to arrive at a reasonable or correct hypothesis quickly. If it cannot, the causal model must be modified to ensure timely and correct diagnoses in the future. The reasoner can recognize that it is missing efficiency information if it arrives at an incorrect hypothesis during diagnosis. Also, if it can build a complete causal chain without outside help during learning then it has enough information to improve its efficiency by including a new association.

4 LBUE - General Process

4.1 Introduction

Learning by Understanding Explanations (LBUE) stems from the notion that if a reasoner can explain a teacher's actions, then s/he can more appropriately generalize what is learned. There is a problem, however, when the reasoner cannot explain such input. The best solution to this problem is to add the information necessary to explain a given case and depend on that new information being applicable in novel but similar cases.

The EDSEL LBUE research specifically explores system understanding of an instructor's hypotheses and subsequent explanations in a diagnostic domain. The system receives the expert's sequence of actions as an input, a partial example is shown in Section 2. When the system is attending to an expert's diagnosis, it must make inferences from the given information to fill in the omitted information. In general, the instructor cannot explain everything at every level of detail. One way to infer the missing information is to attempt causal chaining. When the system receives a symptom, it builds backward chains to all possible findings that could lead to the symptom. When it receives a hypothesis, it builds forward chains to all possible findings that could be caused by the hypothesized fault. If forward and backward chains meet, then the student has an explanation for the hypothesis and thereby has filled in the omitted information. The system can collapse the chain into a fault and symptom pair and save that as a symptom fault set in the mental model. The collapsed chain could then be used more efficiently in future, similar situations.

Often a system will not have enough information to explain the hypothesis. In this case, the instructor's explanation can be useful. First, if the explanation represents a new causal relation, such as X causes Y, it can be added directly to the mental model with high credibility. Second, it may allow bridging a gap to complete a causal chain, thus enabling the system to add a collapsed causal chain as a new association in the causal knowledge, like in EBG. Third, when the explanation does not complete a chain, the system may still infer that the hypothesized fault causes X, and that Y causes the symptom, though there may be intermediate causal links.

As has been noted by many researchers [Mitchell et

al., 1986], EB methods do not generate any knowledge that the system does not already have; rather, existing knowledge is reorganized to be more useful. In collapsing the chains, the LBUE method has some similarities to Explanation-Based Generalization (EBG) [Mitchell et al., 1986] and Explanation-Based Learning (EBL) [DeJong & Mooney 1986]. The student has used available knowledge to form an explanation, which is then stored for later use, somewhat like a macro operator in STRIPS [Fikes et al., 1972]. The LBUE concept goes beyond explanation-based methods by being able to use new information to create a chain when it would not be possible otherwise.

4.2 Algorithm

LBUE, as applied to learning automobile mechanics from examples, requires a very straightforward algorithm. The algorithm must process symptoms, hypotheses, and explanations, and must add to the mental model when possible. An outline of this algorithm follows.

1. From the symptom, chain backward toward possible findings.
2. From each hypothesis, chain forward toward possible effects.
3. If the symptom chain meets a hypothesis chain, then the generalization that *(cause hypothesis symptom)* is added to the symptom fault table and to the mental model.
4. If the chains do not meet,
   (a) Chain backwards from the explanation toward the hypotheses.
   (b) Chain forward from the explanation toward the symptom.
   (c) If both directions can be linked, then the most general relationship *(cause hypothesis symptom)* can be learned.
5. Add explanation to the mental model.

The inputs to the algorithm are the coded protocol and an initial model. The output is an updated mental model and a cognitive trace of the learning process. The following knowledge may be learned:

1. new objects,
2. structural relationships between objects,
3. new causal information, and
4. new symptom fault knowledge.

After this type of learning, diagnosis is more efficient and more powerful for the same or similar problems, because the symptom-fault set can provide better hypotheses more quickly and the mental model is more capable of verifying an explanation.

4.3 Example 1

The general reasoning method is demonstrated by the following two examples. For clarity, the examples will always present a single path through the causal model, rather than the tree-like search that is more typical. The first of these example diagnoses demonstrates learning with just the symptom and hypothesis, without requiring an explanation. The symptom that the instructor reports is slow cranking of the engine. The system chains backward hypothesizing that the crankshaft is spinning slowly, that the starter motor is spinning slowly, that the battery is not generating much electricity:

```
(slow (crank engine)) -->
 (slow (spin crankshaft)) -->
  (slow (spin starter-gear)) -->
   (slow (spin starter-motor)) -->
    (low (contains starter-wire current)) -->
     (low (generate battery electricity))
```

The system achieves this backward chaining by using information about normal function and how, in general, modification of that normal function modifies the rest of the mechanism.

The instructor then suggests the hypothesis that a cracked wire can cause the observed symptom of slow cranking of the engine. The system chains forward, inferring that the battery cable may be carrying less than normal current, that the starter receives less power, and that the wire in the starter receives less electricity:

```
(cracked battery-cable) -->
  (low (contains battery-cable current)) -->
    (low (input starter current))        -->
      (low (contains starter-wire current))
```

In this example, the symptom and hypothesis chains meet, so an explanation is not required. The system understands the instructor's hypothesis, and is able to learn a new symptom-fault set and add the relationship

```
(cause (cracked battery-cable)
       (slow (crank engine)))
```

to the mental model.

4.4 Example 2

A second example will further illustrate the ideas. The instructor may suggest a hypothesis that a cracked distributor cap can cause the observed symptom of the engine cranking but not starting. The system may know, from whatever sources, that for a car to start, the starter must turn, and combustion must occur. The system may further know that for combustion to occur, there must be fuel mixed with air in the cylinder, and a spark from the spark plug. The system might chain backwards from the symptom:

```
(not (start engine)) -->
   (not (combustion cylinder)) -->
      (not (ignite spark-plug))
```

It might not know what a cracked distributor cap can cause, so the system does not understand why it was hypothesized. In other words, the system might not be able to chain anywhere from (CRACKED DISTRIBUTOR-CAP).

The instructor then explains that a cracked distributor cap can allow moisture to collect inside the distributor cap. This causal relationship was not previously known to the student, and it could not have been generated without outside input. The system might be able to complete a causal chain:

```
(cracked distributor-cap)    -->
  (contains distributor-cap moisture) -->
    (low (input spark-plug electricity) -->
      (not (ignite spark-plug))          -->
        (not (combustion cylinder))      -->
          (not (start engine))
```

Now the system understands the hypothesis (and the explanation). It can learn the causal relationship of "cracked distributor cap" causes "engine cranks but won't start". If the student was missing the fact that moisture in the distributor cap can cause less electricity to reach the spark plug, he may still be able to infer that fact based on the explanation having been given by the trusted expert in conjunction with general knowledge about water's effect on electricity. In this case, the explanation not only extends the causal chain, it also assists the system in directing the search for relevant causal relationships.

5 Related Work

Wilkins [1988] uses a somewhat similar method of learning from instruction, that also relies on a model to explain the instructor's problem solving. Both LBUE and Wilkin's approach identify an opportunity to learn by an inability to find such an explanation. A difference is that ODYSSEUS explains actions by linking them to hypotheses while LBUE explains hypotheses by linking them to symptoms. More dramatically, we also learn when we can find an explanation, by collapsing a causal chain into an association or rule.

Other related work includes Mitchell et al's [1985] Learning Apprentice (LEAP) system. LBUE and LEAP are similar in that they both use explanations to properly modify knowledge when given instruction. They differ on how the explanation is related to the knowledge that is learned. LEAP uses two distinct types of knowledge – domain knowledge and implementation knowledge – whereas, LBUE has a single type that is used both for explanation and for performance. LEAP uses its domain knowledge to learn new implementation rules. LBUE allows the system to improve its explanation capability because what is learned can be used in future explanations. A related difference is that LBUE handles incomplete/imperfect domain theories, while LEAP assumes a complete theory.

EBL [DeJong and Mooney 1986] and [Mitchell et al. 1986] also uses explanations derived using a domain model in order to learn new associations. However, it cannot learn any new domain model knowledge. This means that EBL is not appropriate when the causal model is incomplete. Hall's [1986] learning by failing to explain (LBFE) approach was an attempt to remedy this problem. The technique involves isolating the information that is present in the input but is not understood, and subsequently adding it to the existing EBL system. Our approach differs from Hall's work by proposing that the information that must be added in the absence of an explanation is not necessarily explicitly represented in the input. Also, the current effort presents a domain independent notion of learning by attempting to explain that describes how potentially any diagnostic causal model might grow, whereas Hall's effort was, in his own view, domain specific.

Rajamoney and DeJong [1987] have specifically addressed the problem of inconsistencies or missing information in a causal model for simulation. If more than one simulation is possible, their system will experimentally search for disambiguating features in the environment. Although this approach is clearly useful, it does not allow for modification of the general causal information in the model. It concentrates on quantitative values for the current situation, and does not learn any general knowledge. It also does not address improving efficiency of problem solving by generating any associations or rules.

Chandrasekaran and Mittal [1982], used a causal model of a medical domain to generate compiled rules or associations. However, as in EBL, they do not add any truly new knowledge.

6 Conclusion and Future Directions

LBUE is an important tool for automatic knowledge acquisition in diagnostic domains. It takes fuller advantage of an expert's knowledge than some other approaches and uses model-based reasoning to reduce brittleness. LBUE specifically integrates the learning of efficiency information as in EBG with causal model learning.

Protocols can be taken from an expert, coded into a machine readable form and then input to an LBUE process. The system would then be capable of solving the same problem in the future, but would also be able to apply new causal model information to novel but similar cases.

Future directions for LBUE research include extend-

ing EDSEL and applying LBUE to other domains. First, increasing the number of behavioral types that EDSEL is capable of explaining beyond hypotheses to tests will allow the system to learn more about the diagnostic situation. Second, applying the method to other diagnostic domains, possibly a design domain, will help in generating a truly domain independent learning process.

References

Chandrasekaran, B. & Mittal, S. (1982). Deep versus compiled knowledge approaches to diagnostic problem-solving. In *Proceedings of the National Conference on Artificial Intelligence.*

DeJong, G. & Mooney, R. (1986). Explanation based learning: an alternative view. *Machine Learning, 1,* 145–176.

Fikes, R. E., Hart, P., & Nilsson, N. J. (1972). Learning and executing generalized robot plans. *Artificial Intelligence, 3,* 251–288.

Goldstein, I. P. & Bobrow, D. G. (1980). Extending object-oriented programming in smalltalk. In *Proceedings of the 1980 LISP Conference.*

Hall, R. (1986). Learning by failing to explain. In *Proceedings of the National Conference on Artificial Intelligence.*

Johnson, P. (1983). The expert mind: a new challenge for the information scientist. In T. M. Bemelmans (Ed.), *Beyond Productivity.* Netherlands: North Holland Publishing Co.

Lancaster, J. & Kolodner, J. (1987). Problem solving in a natural task as a function of experience. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society.*

Martin, J. & Redmond, M. (1988). The use of explanations for completing and correcting causal models. In *Proceedings of the Tenth Annual Conference of the Cognitive Science Society.*

Minsky, M. (1975). A framework for representing knowledge. In P. H. Winston (Ed.), *The Psychology of Computer Vision.* New York: McGraw-Hill.

Mitchell, T. M., Kellar, R. M., & Kedar-Cabelli, S. T. (1986). Explanation based learning: an unifying view. *Machine Learning, 1,* 47–80.

Mitchell, T. M., Mahadevan, S., & Steinberg, L. I. Leap: a learning apprentice for vlsi design. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence.*

Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess end games. In R. Michalski, J. Carbonell, & T. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach, Volume I.* Palo Alto, CA: Tioga.

Rajamoney, S. A. & DeJong, G. F. (1987). *Active ambiguity reduction: An experimental design approach to tractable qualitative reasoning.* Technical Report UILU-ENG-87-2225, University of Illinois at Urbana-Champaign.

Redmond, M. & Martin, J. (1988). Learning by understanding explanations. In *Proceedings of the 26th Annual Conference of the Southeast Region ACM.*

Wilkins, D. C. (1988). Knowledge base refinement using apprenticeship learning techniques. In *Proceedings of Seventh National Conference on Artificial Intelligence.*

Appendix E
CELIA
The Cognitive Model
The Computational Model
Case Representation
Selected Papers

Redmond, M. (1989). Learning from others' experience: Creating cases from examples. In *Proceedings of the DARPA Case-Based Reasoning Workshop*, Morgan Kaufmann.

Redmond, M. (1989). Combining Explanation Types for Learning by Understanding Instructional Examples. In *Proceedings of the 11th Annual Conference of the Cognitive Science Society.*

Redmond, M. (1990). Distributed cases for Case-Based Reasoning: Facilitating Use of Multiple Cases. In *Proceedings of AAAI-90.*

Redmond, M. (1990). What Should I Do Now? Using Goal Sequitor Knowledge to Choose the Next Problem Solving Step. In *Proceedings of the 12th Annual Conference of the Cognitive Science Society.*

Redmond, M. (1991). Improving Case Retrieval Through Observing Expert Problem Solving. In *Proceedings of the 13th Annual Conference of the Cognitive Science Society.*

In Proceedings of Case-Based Reasoning Workshop (1989)

LEARNING FROM OTHERS' EXPERIENCE: CREATING CASES FROM EXAMPLES [1]

Michael Redmond
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332-0280

ABSTRACT

Cases used for case-based reasoning do not have to be cases that the reasoner itself has solved, or that the programmer has entered. Cases can be acquired by observing expert problem solving. When a reasoner saves a case from its own problem solving, the reasoner already knows what it was trying to accomplish in that episode. In order for a reasoner to use examples it has seen as cases, it must actively process the examples in order to explain and understand the example. The reasoning required in order to acquire a case depends to some extent on how a case is to be represented. Given our theory of case structure, several tasks are necessary in order to interpret the example and store it as a case: inferring the goal behind the observed actions; inferring the relations between pieces; and keeping track of how the actions taken affect the current problem solving context. We have built a system which observes a expert solve a problem in the domain of automobile diagnosis. Among other learning techniques, the system creates cases, in pieces, to be stored for use in later problem solving.

INTRODUCTION

People learn much of what they know from instruction, and the presentation of examples is an important part of instruction in many domains. Students who actively process examples reinforce their knowledge and understanding of the domain principles. They learn how to apply the domain principles. They also may retain details about particular examples, that they can later use to solve other problems. Thus we see that cases used for case-based reasoning do not have to be cases that the reasoner itself has solved.

When a reasoner saves a case from its own problem solving, the reasoner already knows what it was trying to accomplish in that episode. In order for a reasoner to use examples it has seen as cases, it must actively process the examples in order to explain and understand the example. The actions taken by the instructor and seen by the learner do not necessarily map easily and directly into the learner's representation structure for cases. Some inferencing is required in order to interpret the actions and to save them in a form that will be useful for solving future problems.

The reasoning required in order to acquire a case depends to some extent on how a case is to be represented. In our theory, cases are stored in pieces, or snippets [Kolodner 1988]. This allows the reasoner to use small fragments of cases in its reasoning rather than having to wade through large monolithic cases. Each piece is organized around the pursuit of one particular goal, and there are links between the pieces that perserve the structure of the diagnosis. Since the problem solving context changes during the diagnosis, each case piece stores the problem solving context as it was at that time.

Given this theory of case structure, several tasks are necessary in order to interpret the example and store it as a case:

1. Inferring the goal behind the observed actions, in order to break into pieces.
2. Inferring the relations between pieces, that is, the structure of the diagnosis.
3. Keeping track of how the actions taken affect the current problem solving context so that it can be correctly stored in the piece.

We have built a system which observes a expert solve a problem in the domain of automobile diagnosis. Among other learning techniques, the system creates cases, in pieces, to be stored for use in later problem solving. The remainder of this paper discusses how it carries out the necessary tasks listed above, in order to accomplish this objective.

[1] This research was supported by the Army Research Institute for the Behavioral and Social Sciences under Contract No. MDA-903-86-C-173. The author wishes to thank Janet Kolodner for her advice and guidance, and Joel Martin for helpful comments on earlier versions of the paper.

INFERRING INSTRUCTOR'S GOAL

Since case pieces are organized around a goal, before a case can be stored the goals must be known, and the actions supporting the goals must be associated with them. When a system acquires a case from its own problem solving, it knows what goal it is pursuing at a given time. Acquiring a case from somebody else's problem solving, however, is different. Since the instructor's goal is usually not explicitly stated, it must be inferred from his actions. Different goals result in different types of actions being done. The student actively following the example may predict what goal will be pursued next by the instructor. The predicted goal is the first goal considered as a possibility. If the instructor's actions are consistent with that goal, then it is inferred that that is the goal being used. Otherwise, the goal must be inferred bottom up, with all possible goals being possible. This means that if the student gets lost in the example, s/he can find actions that make sense and get back to following along from there, and salvage something from the instructional episode.

In a diagnostic domain some possible goals include generating a hypothesis, testing a hypothesis, interpreting a test, fixing a fault, verifying a complaint, and clarifying a complaint. Figure 1 shows a portion of the instructor's actions in a given example. The complaint had been that the engine stalls, and the instructor has just hypothesized that the fast idle speed is set too low. This hypothesis must be tested. The instructor says that s/he is going to test whether the fast idle speed is low. Then, using his hands, s/he removes the air cleaner. S/he disconnects the radiator fan and connects a tachometer, and otherwise prepares for the test. Then using a specific tool specified in a reference book, s/he carries out the test, reading the value from the tachometer and comparing it to the specifications. These actions should all be saved in a piece for the goal of testing a hypothesis, with the hypothesis being that the fast idle speed is low.

```
(test (low ^fast-idle-speed))
(use (hands))
(do (open ^hood))
(do (remove ^air-cleaner-casing-top))
(do (remove ^air-filter))
(do (remove ^air-cleaner-casing-bottom))
(do (set (position ^gear-shift neutral)))
(use (socket-wrench))
(do (disconnect ^radiator-fan))
(do (connect ^tachometer ^engine))
(do (start ^engine-system))
(do (run ^engine-system) until (temperature ^engine-system warm))
(do (disconnect ^vacuum-advance-hose ^distributor))
(do (plug ^vacuum-advance-hose))
(do (small (open ^throttle)))
(use (c-4812-2c))
(do (connect c-4812-2c ^choke-cam-follower-pin))
(do (release ^throttle-lever))
(ask (desired-fast-idle-rpm nil) ^hood-sticker (reply 2400))
(ask ((rpm ^engine-system) nil) ^tachometer (reply 1600))
```

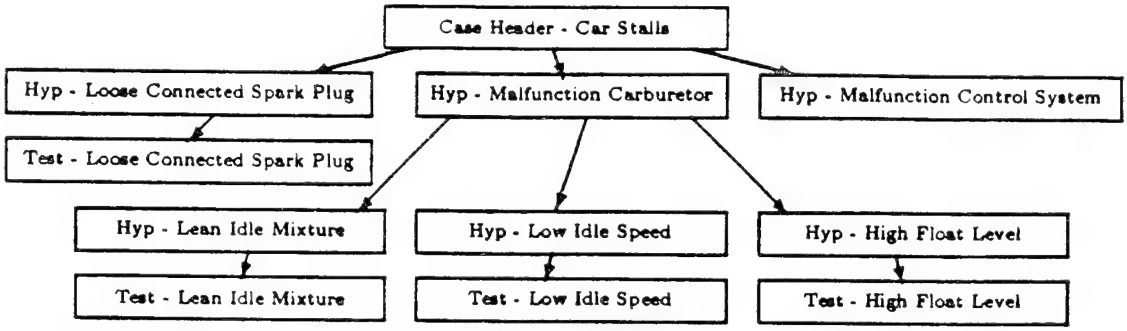
Figure 1: Instructor's Actions to be explained

The representation for goals in our system includes a modifiable, and therefore learnable, structure specifying what types of actions and statements are reasonably expected for fulfilling that goal. Some goals require particular types of actions. Some action types are inappropriate for some goals. Some action types can occur multiple times in the pursuit of a particular goal, some can only occur once. Some action types belong at the beginning of the pursuit of a particular goal, others at the end. To give one example of the type of inference involved, testing a hypothesis *must* include an *ask* type action in order for results to be obtained.

When the expected goal is not pursued, the other known goals must be considered. If none of the diagnosis-specific goals are appropriate a more general goal can be considered, which could result in a diagnosis-specific specialization of the goal being learned. Once the system knows what goal is being pursued, the student can recover and resume following the instructor.

INFERRING PLACE IN CURRENT DIAGNOSIS

We have said that a diagnosis has structure, and that part of acquiring a case from examples is determining that structure. What is this 'structure'? The instructor in most cases diagnoses hierarchically. People doing diagnosis don't hop around between unrelated hypotheses. The experienced mechanic considers a system as a potential source of the problem, then narrows the hypothesis down until a replaceable or fixable unit is determined to be malfunctioning. To a naive observer the hierarchy is not seen, the instructor's actions are sequential, a straight line instead of a tree. People can see this hierarchy easily. The rank novice observed by Lancaster and Kolodner [1987] did not diagnose hierarchically, but the other students, even the one with just six months more experience, did. A system requires knowledge in order to see this hierarchy. It cannot rely on a given pattern of actions from the instructor, but must actually explain or understand what is going on. Figure 2 demonstrates this with an example diagnosis sequence. The top part of Figure 2 shows the structure of the instructor's actions which are shown in the bottom part of Figure 2.



Diagnosis Actions (in order presented)

Hyp - Loose Connected Spark Plug
Test - Loose Connected Spark Plug (Neg.)
Hyp - Malfunction Carburetor
Hyp - Lean Idle Mixture
Hyp - Low Idle Speed
Hyp - High Float Level
Test - Lean Idle Mixture (Neg.)
Test - Low Idle Speed (Neg.)
Hyp - High Float Level
Test - High Float Level (Neg.)
Hyp - Malfunction Control System
Test - Malfunction Control System

Figure 2: Inferred Diagnosis Structure.

Note that a test does not necessarily follow the hypothesis it relates to. Another complication is that there are at least two different reasons that a hypothesis can directly follow another hypothesis - it is a refinement as with the *'lean idle mixture'* hypothesis following *'malfunction carburetor'*, or it is another possibility at the same level, such as with the *'low idle speed'* hypothesis directly following the *'lean idle mixture'* hypothesis. Also note that there is no 'syntactic' cue that the *'high float level'* hypothesis is a refinement of the *'malfunction carburetor'* hypothesis and that the *'malfunction control system'* hypothesis is not. A case representation should store the the case in the structure of the diagnosis, as pictured in the top part of Figure 2. This more accurately reflects the problem solving that occurred in the episode than the linear order shown in the bottom part of Figure 2. This will make the case easier to use in the future.

Knowledge is necessary to understand the hierarchy being used. Causal knowledge and structural relationships from the model are both useful for this process. A hypothesis can go under a previous hypothesis in the hierarchy if it causes the previous hypothesis, if the component involved is part of the previous component, or if the predicate is more refined. We have established a set of heuristics for inferring the relations

between case pieces. The default is that the piece follows from the piece that immediately preceeded it, but this is far from always the case, the other heuristics deal with when that isn't appropriate.

By separating cases into pieces, we have made the system more flexible than it would be with monolithic cases. When the reasoner is trying to acheive a goal it can directly access a piece that previously pursued that goal. When solutions to goals are independent, this makes it easier to find the best solution for each goal. It also means that generalizations involving a particular goal can be more easily created, since information not related to the goal is not confounded with relevant information in the knowledge structure.

At the same time, the cases are reconstructable because the structure of the case is preserved in the links between the case pieces. Therefore, when goals are tightly interconnected, a previous case can be followed as long as it is relevant, by following the links as long as the findings are the same. The diagnostician following such a hierarchically organized case will diagnose hierarchically rather than haphazardly like a novice.

UPDATING THE CURRENT CONTEXT

In order for the stored case pieces to be retrieved when most appropriate, the context when it was appropriate in the expert's judgement should be associated with the piece. The problem solving context includes the initial problem description, any modifications to the problem description, and any relevant actions taken during the current problem solving. That is, in the terms used by Barletta and Mark [1988], it includes the external context and the internal context. Since part of the context changes during problem solving, we have chosen to store the current problem solving context in the piece. In acquiring a case from a solved example, this means that the 'student' must take an active role. The observer must infer how the instructor's actions change the current problem solving context. This is necessary so that the case pieces created will be able to contain the correct problem solving context. Then the piece will be able to be retrieved at the appropriate time in the future, and the correct generalizations will be able to be made. The context must be updated to include hypotheses made and hypotheses ruled out, tests done and their results, and fixes done, plus any changes in the problem description. Most of the information needed can come directly from the observed actions, it just has to be linked to the right slot in the context. However, a student that just passively watches without processing will not get the benefit that the student that does this processing will get.

CONCLUSION

Case-based reasoners in general have used cases that they acquired through their own problem solving and/or cases that were entered by the programmer. An alternative is to acquire cases from solved example problems, such as real students are shown as part of the normal education process. We have described the necessary processing and inferences for acquiring cases from examples in a diagnostic domain. A system has been constructed which uses this and other learning techniques in order to improve its diagnostic abilities. The principles behind the process should transfer to a different type of domain, such as design, but the set of heuristics will probably have to be revised.

References

Barletta, R. & Mark, W. (1988). Breaking cases into pieces. In *Proceedings of Case-Based Reasoning Workshop.*

Kolodner, J. (1988). Retrieving events from a case memory: a parallel implementation. In *Proceedings of a Workshop on Case-Based Reasoning.*

Lancaster, J. & Kolodner, J. (1987). Problem solving in a natural task as a function of experience. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society.*

Redmond, M. & Martin, J. (1988). Learning by understanding explanations. In *Proceedings of the 26th Annual Conference of the Southeast Region ACM.*

Accepted to 1989 Cognitive Science Conference.

Combining Explanation Types for Learning by Understanding Instructional Examples [1]

Michael Redmond
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332-0280
E-mail: redmond@pravda.gatech.edu

Abstract

Learning from instruction is a powerful technique for improving problem solving. It is most effective when there is cooperation between the instructor and the student. In one cooperative scenario, the instructor presents examples and partial explanations of them, based on the perceived needs of the student. An active student will predict the instructor's actions and then try to explain the differences from the predictions. This focuses the learning, making it more efficient. We expand the concept of explanation beyond the provably correct explanations of explanation-based learning to include other methods of explanation used by human students. The explanations can use deductions from causal domain knowledge, plausible inferences from the instructor's actions, previous cases of problem solving, and induction. They involve the goal being pursued and the action taken in support of the goal. The explanations result in improved diagnosis and improved future explanation. This combination of explanation techniques leads to more opportunities to learn. We present examples of these ideas from the system we have implemented in the domain of automobile diagnosis.

1 Introduction

People learn much of what they know from instruction. Presentation of examples can be an important part of instruction. LeFevre and Dixon [1986] found that students prefer examples to written text in learning a procedural task. Reder, Charney and Morgan [1986] found that instruction that included examples was more effective. What is it that makes examples effective teaching instruments?

One characteristic that makes them effective is that active students that try to explain the examples learn through the process of explanation. Lancaster and Kolodner [1988] and Chi, Bassok, Lewis, Reimann, and Glaser [in press] have both observed this in protocol studies. This has been our focus - learning from understanding how a teacher solves an example problem.

Figure 1 summarizes the general process. Essentially, the instructor presents the problem, and appropriate actions or solutions. The student uses various types of knowledge to predict the instructor's actions, and then to understand or explain why the instructor's action or solution is appropriate.

1. The ***instructor*** **states** the problem description.
2. The ***student*** **attempts** to generate an appropriate action for the problem and current context.
3. The ***instructor*** **generates** a correct action or solution for the problem and current context.
4. The ***student*** **then attempts** to explain this action, learning if possible.
5. Continue **with step 2** if the problem is not solved.

Figure 1: General LBUE Algorithm.

The student is testing his ability to diagnose when s/he predicts what the instructor will do. The same techniques s/he would use if s/he were actually diagnosing are used to set up the prediction. In this way, when an opportunity to learn occurs, what is learned will be useful when the student actually goes about diagnosing. The example helps focus the learning.

[1]This research was supported by the Army Research Institute for the Behavioral and Social Sciences under Contract No. MDA-903-86-C-173. The author wishes to thank Janet Kolodner for her advice and guidance, and Joel Martin, Louise Penberthy, and Chris Hale for helpful comments on earlier versions of the paper.

We have constructed a system that creates explanations using deductions from causal domain knowledge, plausible inferences from the instructor's actions, previous cases of problem solving, and induction. The explanations involve the goal being pursued and the action taken in support of the goal. The explanations result in improved diagnosis and improved future explanation. This combination of explanation techniques leads to more opportunities to learn. This paper discusses the different types of explanations, and how they improve future problem solving and explanation.

2 Explanation

In our approach, explanation follows prediction and observation. The first step, therefore, is to compare the prediction with the expert's problem solving for:

- Whether the instructor appears to be pursuing the predicted goal
- Whether pursuit of the goal leads to the predicted action (e.g. the same hypothesis, or the same test of a hypothesis)
- Whether the implementation detail is the same as was predicted (e.g. the same way of carrying out a test or fix)

For each of these, a correct prediction is essentially a successful explanation. Further explanation is required where the prediction isn't met. There can be many different ways of explaining differences. In this paper we discuss explanations involving:

- Inferring the instructor's current goal, and when necessary learning a new goal.
- Inferring the place of the current goal and actions in the diagnosis episode.
- Adjusting the saliency of features for future case retrieval.
- Trying to causally explain actions.

We have also begun to deal with a few other types of explanation that we will not discuss here. For example, explaining differences in implementation detail may rely on differences in car models, available tools, or in the current state of the car.

The types of explanations we make use of overlap with the types of explanations observed by Chi et al [in press]. They observed explanations that:

1. Refine or expand the conditions of an action
2. Explicate or infer different consequences of an action
3. Determine a goal or purpose for an action
4. Give meaning to a set of quantitative expressions.

Their first type of explanation is not a type that we have explored as yet. Our causal chaining explanation type corresponds to their second type, and our inferring the instructor's goal explanation type corresponds to their third type. Their fourth type is not applicable to our domain, though really it is a more specific version of inferring a goal. At a different level, Chi et al [in press] note explanations relating example actions to domain principles and to other example actions. Causal chaining can be seen as relating the observed actions to the domain principles. Inferring the place of the current goal and actions in the current diagnosis episode is one part of relating actions to each other.

In the following sections we will discuss in more detail how explanation of instruction is done, and how it improves the system through what is learned.

3 Inferring Instructor's Goal

Since the instructor's goal is usually not explicitly stated, it must be inferred from his actions. Different goals result in different types of actions being done. The first part of the explanation process is to compare the predicted goal to the instructor's goal, so the instructor's goal must be inferred. The process is focused by the student's prediction of the instructor's goal. The predicted goal is the first goal considered as a possibility. If the instructor's actions are consistent with that goal then it is inferred that that is the goal being used. Otherwise, the goal must be inferred bottom up, with all possible goals being possible. This means that if the student gets lost in the example, s/he can find actions that make sense and get back to following along from there, and salvage something from the instructional episode.

In a diagnostic domain some possible goals include generating a hypothesis, testing a hypothesis, interpreting a test, fixing a fault, verifying a complaint, and clarifying a complaint. Figure 2 shows a portion of the instructor's actions in a given example. The complaint had been that the engine stalls, and the instructor has just hypothesized that the fast idle speed is set too low. This hypothesis must be tested. The instructor says that s/he is going to test whether the fast idle speed is low. Then, using his hands, s/he removes the air cleaner. S/he disconnects the radiator fan and connects a tachometer, and otherwise prepares for the test. Then using a specific tool specified in a reference book, s/he carries out the test, reading the value from the tachometer and comparing it to the specifications.

```
(test (low ^fast-idle-speed))
(use (hands))
(do (open ^hood))
(do (remove ^air-cleaner-casing-top))
(do (remove ^air-filter))
(do (remove ^air-cleaner-casing-bottom))
(do (set (position ^gear-shift neutral)))
(use (socket-wrench))
(do (disconnect ^radiator-fan))
(do (connect ^tachometer ^engine))
(do (start ^engine-system))
(do (run ^engine-system) until (temperature ^engine-system warm))
(do (disconnect ^vacuum-advance-hose ^distributor))
(do (plug ^vacuum-advance-hose))
(do (small (open ^throttle)))
(use (c-4812-2c))
(do (connect c-4812-2c ^choke-cam-follower-pin))
(do (release ^throttle-lever))
(ask (desired-fast-idle-rpm nil) ^hood-sticker (reply 2400))
(ask ((rpm ^engine-system) nil) ^tachometer (reply 1600))
```

Figure 2: Instructor's Actions. The instructor's actions, entered into the system either interactively or by batch in a variable, are predicate forms specifying the type of action, and the action.

The representation for goals in our system includes a modifiable, and therefore learnable, structure specifying what types of actions and statements are reasonably expected for fulfilling that goal. Some goals require particular types of actions. Some action types are inappropriate for some goals. Some action types can occur multiple times in the pursuit of a particular goal, some can only occur once. Some action types belong at the beginning of the pursuit of a particular goal, others at the end. To give one example of the type of deductive inference involved, testing a hypothesis *must* include an *ask* type action in order for results to be obtained. When the expected goal is not pursued, the other known goals must be considered. If none of the diagnosis-specific goals are appropriate a more general goal can be considered, which could result in a diagnosis-specific specialization of the goal being learned. Once the system knows what goal is being pursued, then the same explaining is done as if the goal had been correctly predicted. The student can recover and resume following the instructor.

Figure 3 shows an annotated run of our system CELIA (Cases and Explanations in Learning: an Integrated Approach), reasoning as a student would, realizing that it needs to learn a new goal. For this run of the program we removed knowledge of the goal G-TEST-HYPOTHESIS from the student. This is equivalent to the novice student observed by Lancaster and Kolodner [1987], who came up with a reasonable hypothesis, then proceeded directly to trying to fix it without testing to see if it was a correct hypothesis. The example picks up after the instructor has made the hypothesis that the idle speed is low. The student retrieves a case piece suggesting the repair to do as a prediction of the instructor's actions. The instructor, however, correctly tests the hypothesis. These actions do not match expected action types for carrying out a repair, and in fact are not consistent with action types expected for any of the student's known diagnostic goals. It does, however, on further inspection, fit with expectations for a more general, cross-domain goal, of testing a decision. This enables learning a new diagnostic goal which will be a specialization of the more general goal.

There seems to be a difference between the goals that Chi et al [in press] talk about being inferred and the goals that our system infers. Specifically, if one looks at a goal as a goal type plus

a parameter, our main effort is in inferring the goal type. The goal type would be our goal, for example, G-REPLACE-FIX, and the parameter would be the specific instantiation, for example (INCREASE (POSITION IDLE-SPEED-SCREW)). The parameter comes pretty easily for our system due to the input representation. Chi et al [in press] observed students trying to infer fully instantiated goals where the parameter could be less than obvious. However, the key point is that the student must understand what goal is being pursued in each part of the example as part of explaining the example. Future work can be directed towards inferring the parameter from less well-tailored input.

```
...
Next Task
G-PREDICT-EXPERTS-ACTION

Next predicted goal
G-REPLACE-FIX
 Mentally Simulating strategy  S-RETRIEVE-MEMORY-PIECE  for goal  G-REPLACE-FIX
 retrieve a piece from memory now
 Matches fragments (pieces) -
    (GEN-REPLACE-FIX-LOW-IDLE                7.6000004)
    (GEN-REPLACE-FIX-THERM-COIL-CHOKE    5.6)
    (GEN-REPLACE-FIX-LEAN-CHOKE              5.6)
    (GEN-REPLACE-FIX-TOO-RICH                1.3)
 Simulating based on retrieved piece GEN-REPLACE-FIX-LOW-IDLE
 The fault has been determined to be:  (LOW IDLE-SPEED)
 The fix usually done in previous similar experiences was:  (INCREASE (POSITION IDLE-SPEED-SCREW))
 The method of doing the fix in previous similar experiences was:  ...

Next Task
G-OBSERVE-EXPERTS-ACTION

Expert's next action      ****** NOTE - test if engine is cold when it stalls ******
(TEST (TEMPERATURE ENGINE-SYSTEM (WHEN (STALLS ENGINE-SYSTEM)) COLD))
Expert's next action   :
(DO (DRIVE CAR) UNTIL (STALLS ENGINE-SYSTEM))
Expert's next action   : ****** NOTE - read engine temperature guage when car stalls ******
                          ******       engine is cold when it stalls                  ******
(ASK ((TEMPERATURE ENGINE-SYSTEM) NIL) ENGINE-TEMP-GAUGE (REPLY (COLD)))

Next Task
G-EXPLAIN-DIFFERENCE

 Comparing instructors actions to predicted actions
********** He's using a different goal than expected **********
...
********** don't know the goal being used or know it incorrectly **********
...
He's probably pursuing a specialization of the goal:  G-TEST-DECISION
********** Create that specialization **********
 NEW GOAL:  G-DIAG-TEST-DECISION
**** Add new goal to tables ****
  modify goal-action table
  modify feature-saliency table
  modify goal hierarchy
  modify goal-slot table
  modify slot-action table
  modify slot-context table
  reacting to observing learned goal  G-DIAG-TEST-DECISION
 making new case piece ...   CASE-DIAG-TEST-DECISION-1
...
```

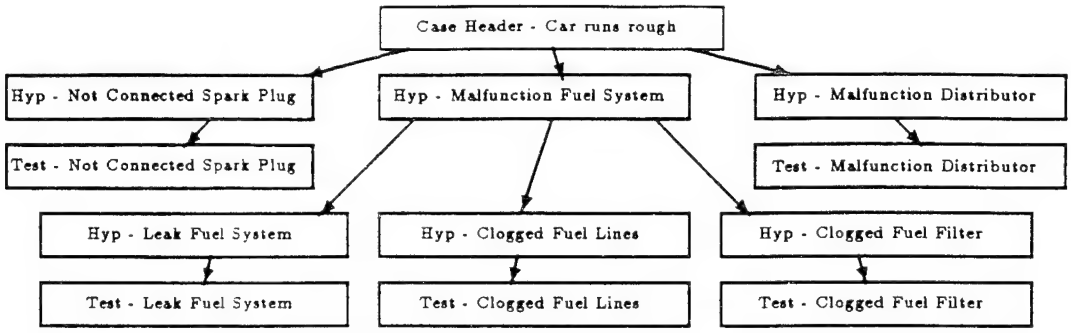
Figure 3: Realizing the need to Learn a Goal.

4 Inferring Place in Current Diagnosis

Inferring the place of the current goal and actions in the diagnosis episode is another step toward understanding observed problem solving. It is not only important in understanding what the instructor is doing, it is also necessary for saving the episode in a useful form as a case for case-based reasoning (CBR) [Kolodner and Simpson 1984]. A case will be more useful in the future if it reflects the problem solving done in the episode.

The instructor in most cases diagnoses hierarchically. People doing diagnosis don't hop around between unrelated hypotheses. The experienced mechanic considers a system as a potential source of the problem, then narrows the hypothesis down until a replaceable or fixable unit is determined to be malfunctioning. To a naive observer the hierarchy is not seen, the instructor's actions are sequential, a straight line instead of a tree. The rank novice observed by Lancaster and Kolodner [1987] did not diagnose hierarchically, but the other students, even the one with just six months more experience, did. The ability to diagnose hierarchically requires knowledge of the hierarchy involved. A system cannot rely on a given pattern of actions from the instructor, but must actually explain or understand what is going on. Figure 4 demonstrates this with an example

diagnosis sequence. The top part of Figure 4 shows the structure of the instructor's actions which are shown in the bottom part of Figure 4.



Diagnosis Actions (in order presented)

Hyp - Not Connected Spark Plug
Test - Not Connected Spark Plug (Neg.)
Hyp - Malfunction Fuel System
Hyp - Leak Fuel System
Hyp - Clogged Fuel Lines
Hyp - Clogged Fuel Filter
Test - Leak Fuel System (Neg.)
Test - Clogged Fuel Lines (Neg.)
Hyp - Clogged Fuel Filter
Test - Clogged Fuel Filter (Neg.)
Hyp - Malfunction Distributor
Test - Malfunction Distributor

Figure 4: Inferred Diagnosis Structure.

Note that a test does not necessarily follow the hypothesis it relates to. Another complication is that there are at least two different reasons that a hypothesis can directly follow another hypothesis - it is a refinement as with the *'fuel system leak'* hypothesis following *'malfunction fuel system'*, or it is another possibility at the same level, such as with the *'clogged fuel lines'* hypothesis directly following the *'leak fuel system'* hypothesis. Also note that there is no 'syntactic' cue that the *'clogged fuel filter'* hypothesis is a refinement of the *'malfunction fuel system'* hypothesis and that the *'malfunction distributor'* hypothesis is not.

Knowledge is necessary to understand the hierarchy being used. Causal knowledge and structural relationships from the model are both useful for this process. A hypothesis can go under a previous hypothesis in the hierarchy if it causes the previous hypothesis, if the component involved is part of the previous component, or if the predicate is more refined.

Chi et al [in press] noted that one type of explanation is relating an action to another action. This process is one way of doing that. It is basically a linking of an action to the action that it follows from, which may *not* be the most recent previous action. The heuristics we use are geared for diagnosis. They were drawn from task analysis of Lancaster and Kolodner's [1987] protocols. They are the set that were necessary to establish the relationships between actions that we saw in the instructor's examples. We don't have any indication whether human students use heuristics such as these to recognize the relationships. Further analysis is required in order to come up with heuristics that would prove useful across domain types, such as for design or planning.

A partial list of heuristics used by our system to explain the instructor's actions in terms of hierarchical diagnosis is shown in Figure 5. The default expectation is that a hypothesis or test will be related to what immediately preceeded it. However, as has been noted, this isn't always the case, and the third, fourth, and fifth heuristics are controls on that. The new action must actually be related to the previous one, by being more specific or causally related. For example, in Figure 4, the hypothesis *'leak fuel system'* is more specific than the hypothesis *'malfunction fuel system'* because the predicate is more specific and the involved component is the same. The hypothesis *'clogged fuel lines'* is more specific than the hypothesis *'malfunction fuel system'* because fuel lines is below fuel system in the partonomy in memory. However the hypothesis *'malfunction distributor'* did not qualify on either count so it had to go in a different place. The third way to satisfy heuristic 3 is for the later hypothesis to be causally related to the previous

hypothesis. The necessity of this is shown by an example. If the hypothesis *'clogged spark plug gap'* follows the hypothesis *'no spark from spark plug'* it would not be placed beneath it because 'clogged' is a different predicate than 'no spark', and isn't more refined. This could easily be a different problem. However, causal knowledge allows linking the one to the other so that the system knows, as a person would, that *'clogged spark plug gap'* is a refinement of the hypothesis *'no spark from spark plug'*.

1. Try to put new hypothesis under most recent previous hypothesis or test.
2. Try to put new test under most recent previous hypothesis.
3. New hypothesis can go under a previous hypothesis if
   - its component is below the previous hypothesis's component in partonomy,
   - if the component is the same and the new predicate is more specific,
   - if the new hypothesis could cause the previous hypothesized fault
4. New hypothesis can go under a previous test if
   - the test showed results indicating abnormal function and
   - the hypothesis is more refined than the test result (component is below the test's component in partonomy or if the component is the same and the predicate is more specific, or if the hypothesis could cause the test result)
5. New test can go under a previous hypothesis if
   - the tested component is the same or below the hypothesis's component in partonomy and the test predicate is the same or more refined than the predicate in the hypothesis,
   - no component in the test is higher than any component in the hypothesis in partonomy
   - or if the tested clause could be a result of the hypothesis")
6. Don't add anything directly under a hypothesis that has already been tested
7. Don't add anything under a test whose results indicated normal function, this should be followed by backtracking
8. Don't add a new test directly under a hypothesis that already has subhypotheses

Figure 5: Heuristics for inferring the structure of a diagnosis.

If the action cannot go after the most recent action then the system must search for its proper place. Many of the other heuristics are limitations on this process, either avoiding potential incorrect placements, or cutting off search that will prove to be unfruitful.

For example, Heuristic 7 allows cutting off search when the instructor would be backtracking. If in Figure 4 the *malfunction fuel system* hypothesis had been followed by a test that showed normal function for the fuel system, then future hypotheses from the instructor should involve other hypotheses that aren't refinements of a fuel system malfunction, and the system can avoid wasted effort by not trying to see if they fit under that hypothesis.

Once the structure of the observed diagnosis has been determined, the case can be stored in memory for use in future problem solving and explanation. The case is stored in pieces so that the particular pieces can be accessed as necessary, and so the representation is flexible enough to handle diagnosis that doesn't have a set pattern of hypotheses and tests. There are pieces for each instance of each goal pursued in the episode. That is, for each hypothesis made, for each test of a hypothesis, for each interpretation of a test, for each fix attempted, there will be a piece. These pieces are linked together to preserve the structure of the case, as inferred in this step. This allows a future diagnosis using the current case to follow the links as long as the findings are the same. The diagnostician following such a hierarchically organized case will diagnose hierarchically rather than haphazardly like a novice. The case pieces, once correctly linked, are stored beneath general knowledge in the model for the car, under related components.

5 Adjusting the Salience of Features

Another important explanation type is adjusting the saliency of features for future case retrieval. It may not seem like adjusting the saliency of features is really explanation. However, when two or more hypotheses are both correct hypotheses, in that they can both cause the observed symptom, causal EBL-like explanations do not provide a way of distinguishing between them. The instructor chooses one of the hypotheses to pursue first. The student predicts a particular hypothesis will be pursued first. If the student's prediction is made based on case based reasoning, then the hypothesis predicted first depends on the matching function. Retrieval of previous cases involves searching for a case or generalization piece which served the goal currently being pursued. The retrieved case piece is selected from the candidate pieces based on a comparison of the feature values of the current problem solving context with the feature values of the problem solving

context at the time of the previous case pieces. So adjusting the matching function by adjusting the importance of features in the problem solving context will lead to the prediction being correct in the future. This is an implicit way of explaining the choice between the hypotheses without having reason to say that one is more likely than the other. The intuition is that such weighting of competitive hypotheses in diagnosis is generally inductive, the mechanic doesn't know for a fact that x fails more often than y, statistics aren't readily available or used, nor can such preference be explained deductively. The weighting is inductive from experience, and from instruction. There is no evidence of this type of explanation in Lancaster and Kolodner's and Chi et al's observations. However, it isn't the sort of thing that would be amenable to study through protocols.

The method of adjusting the saliency of features is fairly simple. It is based on the idea of making features that match when the problem solver is successful more important, and features that match when the problem solver is unsuccessful less important. Since the salience of various features varies depending on the goal being pursued by the problem solver, separate measures of feature importance are maintained for different goals. When the student predicts the same action the instructor makes, the student has been successful. The features of the current problem solving context that matched the features in the previous case are made slightly more important. When the student predicts a different action than the instructor, presumably the student has been unsuccessful. The blame assignment is best made by retrieving another case piece in which the instructor's action was the one done. Figure 6 shows how the blame assignment is done on an example incorrect prediction of a hypothesis. Those features of the current context that more closely match the context of the newly retrieved case piece than the context of the originally retrieved case piece will be made more important. Those features of the current context that more closely match the context of the originally retrieved case piece than the context of the 'correct' piece are made less important. Thus instuction with examples helps deal with the feature saliency problem, by giving feedback on the correctness of case retrieval, allowing comparison of the matching features.

```
Goal - G-GENERATE-HYPOTHESIS
Piece retrieved - (CASE-HYP-CHOKE-THERM 14.280001)
Hypothesis - (MALFUNCTION CHOKE-THERMOSTAT)
Expert's next action - (HYP (LOW IDLE-SPEED))
Piece with hypothesis = (LOW IDLE-SPEED) - (GEN-HYP-ENGINE-STALLS 11.6)
```

Feature	Student's piece	Piece matching Instructor	Feature Importance
CAR-TYPE	Partial match	No Match	less important
CAR-OWNER	Match	No Match	less important
COMPLAINT	Match	Match	no change
FREQUENCY	Partial match	No Match	less important
HOW-LONG	Match	Partial match	less important
OTHER-SYMPT	Match	Match	no change
RULED-IN	Partial match	Match	more important
RULED-OUT	Partial match	Match	more important
TESTS-DONE-W-RESULTS	Partial match	Match	more important
FIXES-DONE	Partial match	Match	more important
CURRENT-HYPOTH	Match (none)	Match (none)	no change
FAULT-DETERMINATION	Match (none)	Match (none)	no change
PARTICIPANTS	Partial match	No Match	less important
LOCATION	Match	Match	no change
WHEN	Partial match	No Match	less important

Figure 6: Example Blame Assignment.

This will lead to the correct piece being retrieved in the same situation in the future. A combination of instruction, case retrieval, and induction has been used to improve the performance of the CBR part of diagnosis.

6 Causal Explanation of Actions

Causal explanations of actions enable filling gaps in the causal domain knowledge through the basic LBUE methods described in Redmond and Martin [1988]. These were an extension of

explanation-based learning (EBL) [DeJong 1983; DeJong and Mooney 1986; Mitchell, Kellar, and Kedar-Cabelli 1986], to allow learning without a complete and consistent domain model. An example will illustrate the ideas. An instructor may present the student with a malfunctioning car in which the engine cranks but does not start. S/he may suggest a hypothesis that the distributor cap is cracked. A complete causal explanation would be:

```
(cracked distributor-cap)      causes
   (contains distributor-cap moisture)    causes
      (low (input spark-plug electricity))   causes
         (not (ignite spark-plug))              causes
            (not (combustion cylinder))            causes
               (not (start engine))
```

If the student can complete the explanation, s/he can learn that a cracked distributor cap causes the symptom of the engine cranking but not starting. If the student was missing some knowledge, it is possible that the knowledge could be inferred as plausible. For example, if the student was missing the fact that moisture in the distributor cap can cause less electricity to reach the spark plug, s/he may still be able to infer that fact based on the partial explanation having been given by the trusted expert, in conjunction with the partial explanation formed by the student and general knowledge possessed by the student about water's effect on electricity.

In addition to enabling filling gaps in the causal domain knowledge, trying to causally explain actions can make causal explanations available as indices to the new case containing the action. Hammond and Hurwitz [1988], and Barletta and Mark [1988] both use this approach, which hasn't yet been implemented in the current system.

7 Conclusion

Explanation of solved example problems is an effective way of learning. A system has been constructed that uses EBL-like deduction, induction, and retrieval of previous cases in creating explanations, improving future diagnoses and future explanations of observed problem solving. The use of multiple types of explanation of examples follows the lead of the studies by Lancaster and Kolodner [1987, 1988] and Chi et al [in press]. Their observations suggest further types of explanation that could be exploited in making our system a better student. The exploitation of instruction turns out to be a powerful way of learning, and integrates several learning techniques.

REFERENCES

Barletta, R. & Mark, W. (1988). Explanation-based indexing of cases. In *Proceedings of a Workshop on Case-Based Reasoning.*

Chi, M., Bassok, M., Lewis, M., Reimann, P., & Glaser, R. (in press). Self-explanations: how students study and use examples to solve problems. *Cognitive Science, in press.*

DeJong, G. & Mooney, R. (1986). Explanation based learning: an alternative view. *Machine Learning, 1,* 145–176.

DeJong, G. (1983). Acquiring schemata through understanding and generalized plans. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence.*

Hammond, K. J. & Hurwitz, N. (1988). Extracting diagnostic features from explanations. In *Proceedings of a Workshop on Case-Based Reasoning.*

Kolodner, J. & Simpson., R. Jr. (1984). Experience and problem solving: a framework. In *Proceedings of the Sixth Annual Conference of the Cognitive Science Society.*

Lancaster, J. & Kolodner, J. (1987). Problem solving in a natural task as a function of experience. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society.*

Lancaster, J. & Kolodner, J. (1988). Varieties of learning from problem solving experience. In *Proceedings of the Tenth Annual Conference of the Cognitive Science Society.*

LeFevre, J. & Dixon, P. (1986). Do written instructions need examples?. *Cognition and Instruction, 3,* 1–30.

Martin, J. & Redmond, M. (1988). The use of explanations for completing and correcting causal models. In *Proceedings of the Tenth Annual Conference of the Cognitive Science Society.*

Mitchell, T. M., Kellar, R. M., & Kedar-Cabelli, S. T. (1986). Explanation based learning: an unifying view. *Machine Learning, 1,* 47–80.

Reder, L., Charney, D., & Morgan, K. (1986). The role of elaborations in learning a skill from an instructional text. *Memory and Cognition, 14,* 64–78.

Redmond, M. & Martin, J. (1988). Learning by understanding explanations. In *Proceedings of the 26th Annual Conference of the Southeast Region ACM.*

To appear in AAAI-90

Distributed Cases for Case-Based Reasoning; Facilitating Use of Multiple Cases

Michael Redmond*
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332-0280
(404) 853-9382
E-mail: redmond@pravda.gatech.edu

Abstract

A case-based reasoner can frequently benefit from using pieces of multiple previous cases in the course of solving a single problem. In our model, case pieces, called *snippets*, are organized around the pursuit of a goal, and there are links between the pieces that preserve the structure of reasoning. The advantages of our representational approach include: 1) The steps taken in a previous case can be followed as long as they are relevant, since the connections between steps are preserved. 2) There is easy access to all parts of previous cases, so they can be directly accessed when appropriate.

Introduction

Case-based Reasoning (CBR) (Kolodner and Simpson 1984) is a method of using previous episodes to suggest solutions to new problems. CBR allows a reasoner to solve problems efficiently when previous similar experiences are available. Problem solving using case-based reasoning usually involves retrieving relevant previous cases, adapting the solution(s) from the previous case(s), if necessary, to solve the new problem, and storing the current episode as a new case to be used in the future.

A case-based reasoner can frequently benefit from using pieces of multiple previous cases in the course of solving a single problem. For example, in protocols taken by Lancaster (Lancaster and Kolodner 1988), mechanics doing a troubleshooting task used pieces of different cases to suggest different hypotheses to consider and tests to perform.

An annotated example from our program CELIA (Cases and Explanations in Learning; an Integrated Approach) (Redmond 1989b), which solves problems in the domain of automobile troubleshooting, illustrates the successful use of multiple cases. In this example, the car is stalling. In forming an initial hypothesis, the reasoner retrieves part of a previous case with the same symptoms that suggests that the idle speed is low and that the mechanic should test whether the engine stalls when it is cold. After carrying out the test the mechanic finds that the engine stalls when warm. Since the results of this test are different than in the previous case, the rest of that case is not useful for further diagnosis.

***** Relevant Case Snippet Retrieved ****
The HYPOTHESIS that Case 101, Snippet Case-Generate-Hypothesis-316 suggests is:
(LOW IDLE-SPEED)
**** Continue with Linked Case Snippet ****
The TEST that Case 101, Snippet Case-Test-Hypothesis-318 suggests is: (TEMPERATURE ENGINE-SYSTEM
(WHEN (STALLS ENGINE-SYSTEM)) COLD)
The TEST-RESULT predicted is:
(TEMPERATURE ENGINE-SYSTEM (COLD))
Result: (TEMPERATURE ENGINE-SYSTEM (WARM))
***** Abandoning Case 101 *****

CELIA recognizes that the first case must be abandoned, and retrieves another case to help it interpret the test result. The new case shares hypotheses and test results with the current situation, rather than just symptoms. It suggests that the problem is a low idle mixture.

***** Continuing with Case 105, Snippet Case-Interpret-Test-505
The RULE-IN that Case 105, Snippet Case-Interpret-Test-505 suggests is: (LOW IDLE-MIXTURE)
The RULE-OUT that Case 105, Snippet Case-Interpret-Test-505 suggests is: (LOW IDLE-SPEED)
...
*** Continuing with Case 102, Snip. Case-Generate-Hypothesis-289
The hypothesis that Case 102, Snippet Case-Generate-Hypothesis-289 suggests is:
(SMALL (DISTANCE THROTTLE-DASHPOT-STEM
THROTTLE-LEVER))
The TEST that Case 102, Snippet Case-Test-Hypothesis-287 suggests is: (DISTANCE THROTTLE-DASHPOT-STEM
THROTTLE-LEVER)
The TEST-RESULT predicted is:
(NOT (SMALL (DISTANCE THROTTLE-DASHPOT-STEM
THROTTLE-LEVER)))
...

It further suggests actions to take to fix the problem (not shown), but after carrying out those actions, the car still stalls. This case, too, is abandoned. Another

*This research has been supported by the Army Research Institute for the Behavioral and Social Sciences under Contract No. MDA-903-86-C-173, and by DARPA contract F49620-88-C-0058 monitored by AFOSR. The author wishes to thank Janet Kolodner for her guidance, Ray Bareiss, and Ashwin Ram for helpful discussions, and Tom Hinrichs, Steve Robinson, Louise Penberthy, and Joel Martin for helpful comments on earlier versions of the paper.

case is retrieved that suggests a new hypothesis: the throttle dashpot is out of place. It suggests checking if the distance between the throttle dashpot and the throttle lever is too short. This hypothesis proves to be correct, the fix it suggests is carried out and the problem is fixed.

The process we are describing is one of using pieces of several cases to solve a problem. We have observed this happening in automobile troubleshooting (Lancaster and Kolodner 1988), medical diagnosis, and meal planning (Hinrichs 1988). We suspect that it is common to any problem solving task that is solved by addressing subgoals individually. It is especially evident when planning and execution are interleaved, where the results of execution are not always as predicted. In order for several cases to be used efficiently in combination with each other, several issues must be addressed.

- How to retrieve a case when some part of it could prove useful.
- How to find and isolate the parts of the previous case that will be useful in the current context.
- How to form generalizations to reflect commonalities in parts of the problem solving experiences.

In this paper we discuss a case representation that enables effective combination of multiple cases by making each part of a case directly accessible, while retaining the links between the parts.

Case Representation

When more than one case is used to solve a problem, frequently only parts of each case will be useful in the synthesis. These parts might be buried within an extended sequence of actions serving many goals. For example, when a reasoner is trying to find a test for a particular hypothesis, the relevant test part of a case is all he needs to focus on. The actions taken and the other hypotheses are not important at that time. Cases must therefore be represented such that their parts can be efficiently accessed.

Traditionally, cases used by case-based reasoners have been treated as monolithic entities [1]. That is, an episode is stored as a single instantiation of a single knowledge structure. Aspects of a case are specified as slots in the representation. Although many of these representations have structured representation within slots and reasoning can be applied to parts of a case, indexing, in general, retrieves the case as a whole.

Treating a situation as a monolithic case and embedding everything in it creates problems for using parts of multiple cases to solve one problem:

- Retrieval of the parts of the case that can be helpful in a given situation has to be a two step process. First, the appropriate case must be accessed, then the currently useful parts hidden inside the case must be found. It can take a lot of effort to find them within the case, even given the right indices to the situation. A one step process allowing direct access to the useful parts of cases would be more efficient.
- Monolithic cases contain too much information for a system to be able to do useful generalization. Since cases have many parts, some of which should be generalized with parts of other cases, but others of which are unique, generalization of commonalities may be delayed.

In order for the appropriate part of a case to be accessed when it can be useful, in the middle of problem solving, it is advantageous to divide cases into pieces. In our model,

- Cases are stored in pieces, or *snippets* (Kolodner 1988). This allows the reasoner to use small fragments of cases in its reasoning rather than having to wade through large monolithic cases.
- Each snippet is organized around one particular goal, and contains information pertaining to the pursuit of that goal.
- Each snippet contains the current problem solving context at the time the goal was initiated, including the initial problem description and results of actions taken so far.
- There are links between the snippets that preserve the structure of the diagnosis. Each snippet is linked to the snippet for the goal that suggested it and to the snippets for the goals it suggests.

Content of Snippets

Each snippet can be thought of as a scene of the larger episode. A snippet has three main types of information. First, is the problem solving context at the time of the snippet's occurrence. Second is information related to the goal that the snippet is centered around. Last is information linking the snippet to other related snippets. Figures 1, 2, and 3 together comprise an example of the internal representation of a snippet representing the goal of testing the hypothesis that the carburetor float bowl had too high a fuel level.

Context. Problem solving context includes actions and results of actions taken earlier in the problem solving, as well as features of the problem. *Global context* is the features given for the overall problem situation, *internal context* is the circumstances, state or knowledge established by the actions already taken as part of the problem solving. In the automobile troubleshooting domain, internal context includes tests that have been done and their results; information or hypotheses that have been ruled out or ruled in during problem solving; fixes that have been made during problem solving; and the current hypothesis. The global context includes the chief complaint; other symptoms; how frequently the symptoms occur; how long the problem has been going on; any particular ambient temperature range when failure occurs; any particular weather conditions when failure occurs; the car model; the customer; the mechanics involved; and where and when the problem solving occurred. Global context remains the same across snippets of a case, but internal context changes.

[1] See Related Work section for some exceptions.

Figure 1 shows an example of the problem solving context part of a snippet.

CASE-TEST-HYPOTHESIS-130

CONTEXT

```
Internal
  Ruled-In     (Lean (Position (Idle-Mixture-Screw)))
  Ruled-Out    (Low (Position (Idle-Speed-Screw)))
               (Lean (Position (Idle-Mixture-Screw)))
               (Incorrect (Position (Throttle-Dashpot)))
  Tests-Done-N-Results
        (Temperature Engine-System
              (When (Stalls Engine-System)) Cold)
           (Hot (Temperature Engine-System))
        (Stalls Engine-System)
           (Stalls Engine-System)
        (Small (Distance Throttle-Dashpot-Stem Throttle-Lever))
           (Not (Small
                 (Distance Throttle-Dashpot-Stem Throttle-Lever))
  Fixes-Done   (Increase (Position (Idle-Mixture-Screw)))
  Solution: NIL
  Current-Hypoth   (High (Contains Carburetor-Float-Bowl Fuel))

Global
  Complaint          (Stalls Engine-System)
  Other-Sympt        (Rough (Run Engine-System))
  Frequency          Weekly
  How-Long           2months
  Amb-Temp-G-Fail    Any
  Weath-G-Fail       Rainy
  Car-Type           (1981 Ford Granada)
  Car-Owner          Davis Cable
  Participants       Mark Graves, David Wood
  Location           Mikes-Repair-Shop
  When               2843569149
```

Figure 1: Example Case Snippet Context.

The problem solving context of a snippet is used for matching during retrieval. As Barletta and Mark (1988) have suggested, both internal and global problem solving context are necessary to maintain coherence and consistency of actions. Since snippets include both internal and global problem solving context, retrieval results in *usefully* similar case pieces.

This form of context creates advantages for combining multiple cases to find a solution. When there is a need to access part of another case, having the internal context available allows matching on results of previous problem solving. Thus a snippet which followed from similar steps and results can be favored. In this way both access issues are addressed: accessing a case that is relevantly similar, and accessing the part of the previous case that will be useful in the current context. An appropriately relevant snippet can be directly accessed. With monolithic cases the internal context at each point in the problem solving would not be available to make accessing parts easy. By saving the context with each piece we are trading space for flexibility. Any method, in order to be as flexible, would have to either represent the internal context at each point, or be able to recompute it at retrieval time, an expensive proposition.

In addition, though not currently implemented, representing the internal context enables analytical reasoning that could determine that the current context is incompatible with something already done prior to the snippet in the previous situation, thus averting failure.

Pursuit of Goal. Each snippet is centered around the pursuit of one goal. It is here that the actions taken in pursuit of a goal and the results of those actions are recorded. When a snippet is retrieved during problem solving, these slots suggest the actions to take and the results to expect if the situation is the same as in the previous case. We have identified 7 types of goals involved in troubleshooting, Table 1 lists those goal types and their associated slots. Figure 2 shows the slots for a test: the test that was done, the method of carrying it out, the tools used, and the result. In general, the goal-related part of a snippet needs to include the actions carried out to achieve the goal and the effects of the actions.

CASE-TEST-HYPOTHESIS-130

PURSUIT OF GOAL

```
Goal        G-TEST-HYPOTHESIS
Test        (High (Contains Carburetor-Float-Bowl Fuel))
Test-Method
        (Turn-Off Engine-System)
        (Remove Carburetor-Air-Horn-Screw)
        (Remove Carburetor-Air-Horn)
        (Ask (Level Fuel Carburetor-Float-Bowl)
                    Scale-On-Carburetor-Float-Bowl)
Test-Tools      Screw-Driver
Test-Result     (High (Level Fuel Carburetor-Float-Bowl))
```

Figure 2: Example Case Snippet Pursuit of Goal.

Goal Type	Slots for Goal Type
Generating a hypothesis	hypothesis generated
Clarifying a complaint	actions taken, information gathered
Verifying a complaint	actions taken
Testing a hypothesis	test that was done, method of carrying it out, tools used, result
Interpreting a test (either of hypotheses or repairs)	hypotheses ruled out, hypotheses suggested or ruled in
Making a fix or replacement	replacement or fix done, method of carrying it out, tools used
Testing a fix or replacement	test that was done, method of carrying it out, tools used, result

Table 1: Goal Types and Their Slots.

Linkage. While it is important to divide cases into snippets so that parts of cases can be easily and directly accessible, it is also important to be able to reconstruct the case. Sometimes a number of steps in the same case can provide useful guidance. A hypothesis suggests a test, the result suggests an interpretation, the interpretation suggests a fix, and the fix is associated with a test of the fix. As long as the expectations from a previous case are upheld in the new situation, the reasoner can benefit by following the sequence of reasoning steps from the recalled case. In order to en-

able such reconstruction, snippets include links to the snippets for the goals they follow from and the goals that follow from them. Briefly, the idea is, first, retain the links so that in the future use of the snippet, the step that it suggested will be suggested. Second, save the value of the main slot of the preceding snippet to facilitate making generalizations involving the previous step taken. Note that the portion of the snippet shown in Figure 3 has a slot for the previous *hypothesis* because it was preceded by a generate hypothesis snippet, whose main slot was a hypothesis.

CASE-TEST-HYPOTHESIS-130

LINKAGE

Link-Down	CASE-INTERPRET-TEST-140
Link-Up	CASE-GENERATE-HYPOTHESIS-125
Prev-Hypothesis	(High (Contains Carburetor-Float-Bowl Fuel))

Figure 3: Example Case Snippet Linkage.

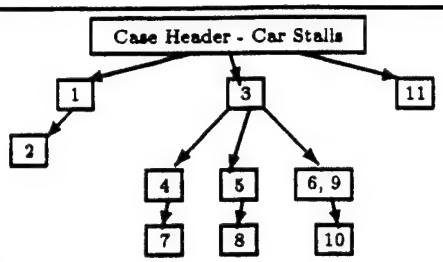
Links Between Snippets

As mentioned above, snippets are linked together in a manner that preserves the underlying structure of the goals pursued in the case. Figure 4 shows the linkages between snippets for a case of a stalling car. This case illustrates the differences that can occur between temporal order and the underlying structure of which snippets follow from which. Each node represents a case snippet for a particular goal pursued in the case. Each link represents a relationship between goals in the case. Arrows point from a snippet to the snippets that it suggests.

Preserving these intra-case links is important for future use of the case. For example, in diagnosis, the case structure preserves which hypothesis suggested a test, and what test result suggested a hypothesis, even if they were not contiguous in the processing. To demonstrate this, in Figure 4, step 6 follows from step 3, since it is a refinement of the previous hypothesis. Step 7 follows from step 4, since it is a test of the hypothesis. Step 11 does not follow from any of the previous hypotheses or tests, it is the start of a new direction after a dead end was reached. This means that the important structure in a case is not the temporal order of the actions taken. Rather, it is which goals follow from which other goals.

For example, in automobile troubleshooting, a hypothesis that the alternator belt is loose could follow from a hypothesis that the battery is not charging since it is a potential root cause. Or a hypothesis that the radiator hose is leaking could follow from a hypothesis that there is a leak in the cooling system, since it is a narrowing of a hypothesis.

Preserving this linkage enables a case to provide guidance as to what to pursue next as long as the results continue to be the same as in the previous case. In troubleshooting, this allows retrieval of a case fragment with a particular hypothesis to yield a connection to the test to do next. The guidance is not affected by any idiosyncratic temporal ordering that does not reflect the underlying structure of the previous case.



Diagnosis Actions (in order presented)

1. Hyp - Loose Connected Spark Plug
2. Test - Loose Connected Spark Plug (Neg.)
3. Hyp - Malfunction Carburetor
4. Hyp - Lean Idle Mixture
5. Hyp - Low Idle Speed
6. Hyp - High Float Level
7. Test - Lean Idle Mixture (Neg.)
8. Test - Low Idle Speed (Neg.)
9. Hyp - High Float Level (restate)
10. Test - High Float Level (Neg.)
11. Hyp - Malfunction Control System

Figure 4: Case Structure.

Snippet Access

Snippets can be accessed two ways:

- Directly, through retrieval, matching the current situation to the snippet's goal and context.
- Sequentially, by following links between snippets.

In CELIA, retrieval via direct access is first restricted to snippets that are centered around the goal type being considered. Then a weighted similarity metric is used, with matching occurring for all features within both the internal and global context. An empirical adjustment of the weight on a feature's importance is made based on the success or failure of prediction during learning (Redmond 1989b). As would be hoped, the similarity metric quickly comes to favor many of the features in the internal context, and give less importance to features in the global context that seem spurious, such as the car owner, the participants, and the location. These, however, are not eliminated, so they can play a part in some unusual situation in which they are important. Further work will investigate using some form of explanation based indexing (Barletta and Mark 1988) in conjunction with this.

Retrieval by sequential access is easy given our case representation. Snippets have links to other snippets that follow from them. When a snippet has been used to suggest an action, if the result is the same as in the snippet's execution, the reasoner can follow the link to the next goal and its actions. Retrieval by sequential access is favored over direct access when it is appropriate. This retains coherence and avoids unnecessary processing.

Advantages of the Approach

Our system, CELIA, learns from observing an expert's actions (Redmond 1989b, 1989a). It uses parts of multiple cases for two tasks

- To predict and explain the instructor's actions as it is learning. The experience is then saved in the distributed form we have described here. Redmond (1989c) explains how this distributed case representation can be constructed from observing raw input.
- To provide guidance during problem solving.

The distributed case representation has advantages for both processes. Here we discuss the advantages for problem solving.

- There is easy access to all parts of previous cases, so they can be directly accessed when a snippet's goal is to be pursued.
- The structure of the case is retained so it can be reconstructed as a whole or in part when necessary. The actions taken in the previous case can continue to provide guidance as long as the situation remains usefully similar to that of the previous case. Operationally, in the current system this means as long as the same results are obtained in the step as in the corresponding step in the previous case.
- Generalizations of case snippets can be formed for the pursuit of a particular goal, but not hindered by the pursuit of other goals in the cases.

While empirical measures of the learning part of the system have been made, our evaluation of the representational approach is based on the fact that it enables, without much cost, flexible problem solving that would otherwise be difficult. With monolithic cases, not only might the reasoner not have the right indices for the case, finding the right part of a case to use for the current task situation is effortful. Even if the case is indexed so that it will be accessed when any of its parts are relevant (*e.g.* by important features of the internal context at each and every point during the case – the tests done and their results, the fixes done ...), once that case is accessed it is necessary to find the point in the case's problem solving where its context best matches the current context. Making this process as simple as CELIA's process would require including all the same information that CELIA's cases have – the context at each point during the case (for indices), directly associated with the step in the case. This case representation content would not be distinguishable from our theory except that our representation preserves the underlying order of the steps.

The ideas discussed raise some issues. First, as snippets become smaller, the distinction between cases and situation/action rules may seem to blur. If problem solving does not benefit from the overall context provided by a whole case, then the case representation might be equivalent to individual decision rules. A related question one might ask is whether problem solving behavior constructed from relatively local decisions can be globally consistent, or will unforeseen interactions between goals creep in. An advantage of traditional case-based reasoning techniques is that a whole case has a coherency that holds together the problem solving behavior. We do not want to lose coherency when we break cases into pieces. We have discussed a number of important ways in which our snippet representation differs from decision rules. First, the links between actions are retained, so a case can be followed as long as the results of actions are as predicted by the previous case. Sequential access of snippets leads to a goal-directed coherence that is not inherent in a set of individual decision rules. Second, snippets include in their context both the initial problem description and the results of actions taken up to that point. This means that when direct access is used the choice of actions to take is directly influenced by the problem solving that has already occurred. Third, the case snippet can provide useful suggestions even when there is only a partial match to the current situation (*i.e.* when a rule may not apply).

Another issue we must address is what size snippet is most useful for problem solving. There is a trade-off between the efficiency of being able to match and use a single case as the solution to the problem, and the generality of matching to the parts that are applicable in the current situation. We have suggested that the appropriate division is for each snippet to concern a single goal. We should clarify that this means that they are concerned with a *leaf* goal. A high level goal such as *explain anomaly* would be broken down into subgoals using knowledge of goals and subgoals. The lowest level goals are the ones that the reasoner would look for guidance in achieving. It is this level of goals that the case snippets are organized around. Applying this to the advantages suggested above, the approach has these advantages:

- Direct access is to the parts of previous cases involved in the pursuit of the types of goals that the problem solver seeks guidance in achieving.
- When generalization of case snippets is added, it can be done for the pursuit of a particular low-level goal for which the reasoner might later want guidance.

It might be argued that the number of indices necessary increases as the cases are divided into smaller chunks and each chunk requires indices. However, if parts of the larger chunks are to be accessible, then equivalent numbers of indices are necessary in order to be able to get an indication that some part of the larger chunk would be of value in the current circumstances. Thus, snippet size and number of indices are independent of each other.

This division of cases into multiple snippets based on goals pursued is important when multiple cases are used to solve different parts of a problem. When changes lead to the need to access a different case to get help pursuing a goal, the part of the case that should be accessed is available such that it can be found in a timely manner. Such division would not be important in a domains in which cases can *only* be considered as a whole (as in *e.g.* HYPO (Ashley & Rissland 1987)).

Related Work and Conclusions

Most CBR approaches have represented cases as single units and reasoned based on one case. MEDIATOR (Simpson 1985) made use of parts of multiple cases in coming to a solution. However, MEDIATOR had to first choose a case, then access the relevant part. More recently, several CBR approaches have separated cases into pieces.

JULIA's (Kolodner 1989; Hinrichs 1988) case pieces represent scenes that are related partonomically and taxonomically. Its snippets, like CELIA's, facilitate synthesizing parts of multiple cases to form a solution, and effective generalization. JULIA does not need the type of links used in our representation, however, because of the relatively low amount of difference in structure of cases, and a relatively static set of goals across problems, with limited need to pursue them in any particular order.

Derivational Analogy (Carbonell 1986) is somewhat similar to our approach in that it saves the problem trace, including generation of subgoal structures and generation of alternatives. A key difference is that our approach provides both direct and sequential access to parts of the problem solving. Derivational Analogy only accesses a trace at the beginning of a problem. If a case can no longer provide guidance a different case must be accessed from the top and the reasoning followed from there. The start of the problem trace is accessed when it shares a subgoal chain with the current situation. Therefore, derivational analogy cannot retrieve a case or part of a case with a similar current subgoal but a different way of getting to it.

Barletta and Mark (1988) group their cases into pieces such that the actions that are used to recover from each of the hypothesized faults are in the same piece. This serves part of the purpose of dividing the cases into pieces, direct access to relevant actions. It appears to be specific to the particular goal of fixing a known fault, however. It is not as flexible as organization by goals, in that it serves a particular goal type, fixing a problem, but does not allow easy direct access to all the goals pursued in the previous case.

Kopeikina, Bandau, and Lemmon (1988a, 1988b) suggested the need for cases that represent how a situation develops over time. Their approach was to divide a case into problem description; action plan; description of effect of implemented action; description of the 'no need for treatment state'; remove controls; and description of a 'no problem state'. Such a case is always accessed at the beginning, and all of the main actions are within the second piece. If the results are not as expected, another case can be used to recover from that *new* problem, however, it is accessed from the beginning as a whole case. They argue against dividing cases up into unconnected individual cases, but do not consider an approach such as ours. Our use of snippets which retain both the internal context, and the internal structure of the case addresses this need.

The use of parts of multiple cases, and the division of cases into linked, goal-centered fragments provides flexibility to recover from changes or unexpected results, while retaining goal-driven processing. The case representation enables direct access to usefully similar parts of previous cases, while retaining the opportunity to follow significant portions of a previous case. When generating a multi-step solution and the solution can be synthesized from multiple cases, our distributed case representation provides significant flexibility advantages. This is important in numerous domains, including automobile troubleshooting, medical diagnosis, many design problems, and we suspect any problem solving task that is solved by addressing subgoals individually.

References

K. Ashley and E. Rissland 1987. Compare and contrast, a test of expertise. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-87)*, San Mateo, CA. Morgan Kaufmann.

R. Barletta and W. Mark 1988. Breaking cases into pieces. In *Proceedings of Case-Based Reasoning Workshop*, St. Paul, MN.

J. Carbonell 1986. Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach, Volume II*. Morgan Kaufmann, Los Altos, CA.

T. R. Hinrichs 1988. Towards an architecture for open world problem solving. In *Proceedings of a Workshop on Case-Based Reasoning*, San Mateo, CA. Morgan Kaufmann.

J. Kolodner and R. Simpson Jr. 1984. A case for case-based reasoning. In *Proceedings of the Sixth Annual Conference of the Cognitive Science Society*, Hillsdale, NJ. Lawrence Erlbaum Associates.

J. Kolodner 1988. Retrieving events from a case memory: a parallel implementation. In *Proceedings of a Workshop on Case-Based Reasoning*, San Mateo, CA. Morgan Kaufmann.

J. Kolodner 1989. Judging which is the "best" case for a case-based reasoner. In *Proceedings of the Second Workshop on Case-Based Reasoning*, San Mateo, CA. Morgan Kaufmann.

L. Kopeikina, R. Bandau, and A. Lemmon 1988a. Case-based reasoning for continuous control. In *Proceedings of a Workshop on Case-Based Reasoning*, San Mateo, CA. Morgan Kaufmann.

L. Kopeikina, R. Bandau, and A. Lemmon 1988b. Extending cases through time. In *Proceedings of Case-Based Reasoning Workshop*, St. Paul, MN.

J. Lancaster and J. Kolodner 1988. Varieties of learning from problem solving experience. In *Proceedings of the Tenth Annual Conference of the Cognitive Science Society*, Hillsdale, NJ. Lawrence Erlbaum Associates.

M. Redmond 1989a. Combining case-based reasoning, explanation-based learning and learning from instruction. In *Proceedings of the Sixth Annual International Workshop on Machine Learning*, San Mateo, CA. Morgan Kaufmann.

M. Redmond 1989b. Combining explanation types for learning by understanding instructional examples. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, Hillsdale, NJ. Lawrence Erlbaum Associates.

M. Redmond 1989c. Learning from others' experience: creating cases from examples. In *Proceedings of the Second Workshop on Case-Based Reasoning*, San Mateo, CA. Morgan Kaufmann.

R. L. Jr. Simpson 1985. *A Computer Model of Case-Based Reasoning in Problem Solving*. PhD thesis, Georgia Institute of Technology, Atlanta, GA.

Cognitive Science Conf. 1990

What Should I Do Now? Using Goal Sequitur Knowledge to Choose the Next Problem Solving Step [1]

Michael Redmond
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332-0280
E-mail: redmond@pravda.gatech.edu

Abstract

Many problems require multi-step solutions. This is true of both planning and diagnosis. How can a problem solver best generate an ordered sequence of actions to resolve a problem? In many domains, complete pre-planning is not an option because the results of steps can vary, thus a large tree of possible sequences would have to be generated. We propose a method that integrates the use of previous plans or cases with use of knowledge of relationships between goals, and the use of reasoning using domain knowledge to incrementally suggest the actions to take. The suggestion process is constrained by heuristics that specify the circumstances under which an instance of a particular reasoning goal can follow from an instance of other reasoning goals. We discuss the general approach, then present the suggestion methods and the constraints.

1 Introduction

There are many problems for which a multi-step solution must be generated. Such problems occur both in planning and in diagnosis. For instance, in automobile troubleshooting, a possible sequence of actions includes clarifying the complaint, verifying the complaint, generating hypotheses in some order, testing hypotheses, interpreting the test results, carrying out repairs, and testing the repairs. Complete pre-planning of troubleshooting steps may be inefficient. The number of possible choices and the variety of possible results of the actions can lead to a large, very bushy tree of possible paths. Generation of the complete trouble-tree for the given car and problem would be an expensive task to do, and might not even be possible. In addition, one action may not directly follow from the previous action. The question raised is how to best generate an ordered sequence of actions to resolve a problem.

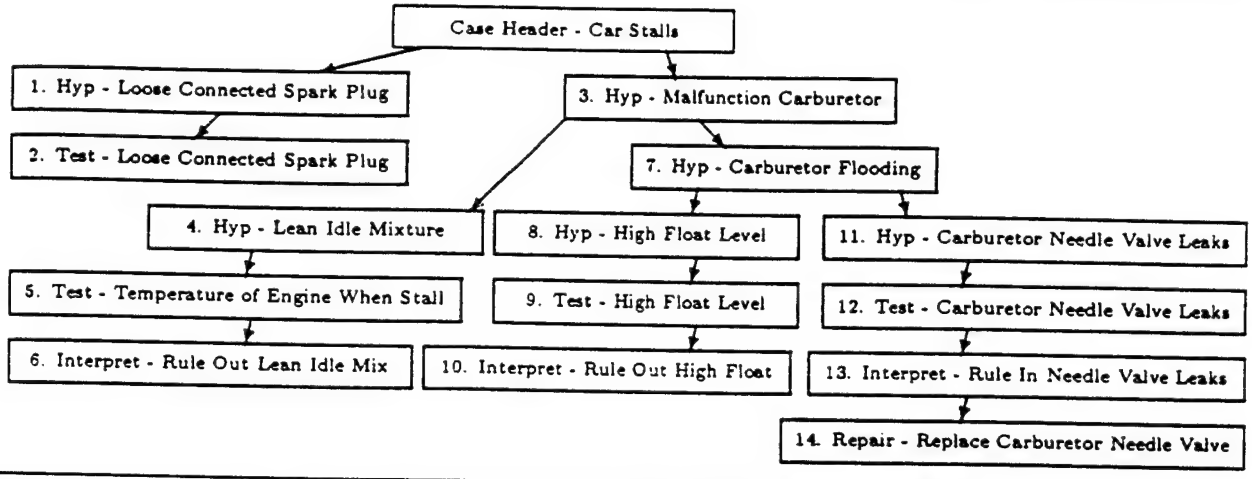
For example, for a stalling car, a possible sequence of actions is shown in Figure 1. The problem solver first hypothesizes a loose spark plug and a test of the hypothesis finds it not to be true. The problem solver hypothesizes that the carburetor is malfunctioning, then refines that guess to the more specific hypothesis of the idle mixture being lean. This is tested and the result suggests that the idle mixture is probably not the problem. Next, the problem solver generates a hypothesis that the carburetor is flooding, refines that to a hypothesis that the float level has become set too high, and tests for that. However, the test result indicates that that is not the problem. The problem solver generates another refinement, that the carburetor needle valve is leaking, allowing fuel in when it should not. This is tested, and is found to be true. A repair is done, and is tested, and results in the elimination of the problem.

These actions are not independent. Results of early actions influence future actions, and results cannot be predicted with certainty. If the spark plugs turned out to be loose, a repair would be done at that point and the problem solving would be complete if that is the only problem. If the needle valve is not leaking, further steps would be necessary beyond those in Figure 1. If, as a by-product of the test of the float level being high, it was determined that the fuel level in the carburetor was not too high, a different hypothesis would have been pursued opportunistically, instead of continuing to pursue the carburetor flooding hypothesis.

This example illustrates several points. First, it shows a situation where complete pre-planning would be inefficient. Not only can the number of possible choices and the variety of possible results of the actions lead to a large, very bushy tree of possible paths, but in many problems much of the tree would not be used. Second, it illustrates that when choosing the next action to take, the next action may not follow from the

[1] This research has been supported by the Army Research Institute for the Behavioral and Social Sciences under Contract No. MDA-903-86-C-173, and by DARPA contract F49620-88-C-0058 monitored by AFOSR. The author wishes to thank Janet Kolodner for her advice and guidance, and Tom Hinrichs, Steve Robinson, and Joel Martin for helpful comments on earlier versions of the paper.

most recent action. For example, step 7 follows from step 3. In addition, following a trouble-tree would not suggest taking advantage of unexpected opportunities, such as following up of the by-product of a test, as seen above.



The nodes represent the different goal instances that have been pursued. They are numbered in the temporal order in which they occurred. The links represent the relationships of which goal instance followed from which goal instance.

Figure 1: A Multi-step Solution in Automobile Troubleshooting.

The problem to be addressed in this paper is three-fold:

1. How can a problem solver efficiently generate successive goals and actions in a multi-step solution?
2. What knowledge is needed to generate the succeeding goals and actions?
3. How should the generation process be controlled and suggestions selected?

We address this problem in the task domain of automobile troubleshooting. Our program, CELIA (Cases and Explanations in Learning; an Integrated Approach), solves problems by generating and achieving reasoning goals. As in the example above, often later goals cannot be generated until earlier ones have been achieved. In addition, the program learns by understanding and explaining the statements and actions of a teacher.[2] The same process that generates goals during problem solving generates them during learning, where they act as expectations of what the teacher will do or say next.

As we will show, our approach integrates the use of four important types of knowledge to generate goals: previous solutions or cases (as in *Case-based Reasoning* [Kolodner and Simpson 1984]), knowledge of relationships between reasoning goals (in this case, knowledge of the troubleshooting process), and causal knowledge, including structural and functional knowledge of the domain. We will discuss generation of suggested actions, and control of the process, and then present an example.

2 Generation of New Subgoals and Actions

We have found four types of knowledge useful for generating subgoals and actions:

1. Case knowledge.
2. Causal knowledge of components, mainly functional knowledge.
3. Structural knowledge of component parts, including part/whole and adjacency relationships.
4. Knowledge of how to do the task, or the relationships between reasoning goals.

[2] Not natural language.

Case Knowledge: Case-based Reasoning (CBR) is a method of using previous episodes to suggest solutions to new problems. CBR is an important problem solving technique because it allows a reasoner to solve problems efficiently when previous similar experiences are available and complete knowledge is not present. In this type of problem, a case provides an ordered set of actions that have worked in the past. Thus remembering a part of a case, or *snippet*, suggests the next action. A next action can be suggested by the case the reasoner is currently reasoning from, or if context changes, by another case that becomes more relevant.

Causal knowledge: A next action or subgoal can also be suggested by causal knowledge. A causal link from a previous action can make the suggestion. Some aspect of the previous goal, for example the test done, the test result, the hypothesis generated, the fix done, something ruled out, is the starting point for the reasoning. The reasoning can proceed from that point forward toward effects of that aspect, or backward toward causes of that aspect.[3] The furthest point of progress in the causal reasoning is suggested as the value for the instance of the reasoning goal to be suggested. The causal reasoning uses both functional and structural knowledge of the domain.

Structural knowledge of components: Part/whole and topological knowledge of components involved in a previous action can suggest the next action. For example, a problem with the electrical system might be due to a problem with the battery. If the most recent instance of a generate hypothesis goal was that the electrical system is faulty, the next appropriate goal instance might be the generation of a hypothesis that the battery is faulty.

Knowledge of how to solve problems in the domain: How problems are normally solved is also important to generating goal and action sequences. Our method uses heuristic knowledge in the form of a set of the types of reasoning goals, or *goal types*, which can follow from each reasoning goal type, called *goal sequitur* knowledge, or sequitur knowledge for short.[4] [5] For example, hypothesis generation goals can be followed by tests of hypotheses, or by further hypothesis generation.

In general, there are many possible succeeding goals a problem solver might generate at any time. Good problem solvers generate goals that can lead them toward their final destination in the most opportune way. In the remainder of this paper we will present a way to choose the next goal or step wisely. We will show that the fourth type of knowledge listed above, knowledge of the problem solving task, is primary to this endeavor, providing guidance for moving toward a solution.

Our method uses three main types of heuristics for this task: *suggestor* heuristics suggest new steps, *restrictor* heuristics constrain the behavior of the suggestors, and *selector* heuristics choose the best of the suggested next steps.

We begin by presenting the four main types of suggestor heuristics:

1. Case Sequential Access
2. Case Direct Access
3. Causal link
4. Refinement (Part/Whole)

The suggestor heuristics, or suggestors, indicate ways to generate possible instances of the consequent goal type.[6] Running these heuristics results in a set of possible succeeding reasoning goals and actions.

[3] Reasoning is uncoupled from the reasoning goal involved. Given a hypothesis that the fuel mixture is too lean, reasoning proceeds from the state that the fuel mixture is too lean (which may or may not be true), not from the *hypothesis* that the fuel mixture is too lean. Thus the causal reasoning does not depend on the reasoning goals involved in the domain, or vary depending on those involved.

[4] *Goal types* are general types of reasoning goals such as clarifying the complaint, verifying the complaint, generating hypotheses, testing hypotheses, interpreting the test results, carrying out repairs, and testing the repairs. They could also be considered subtasks of troubleshooting. Goal instances are specific instantiations of goal types, such as the specific test used to test a specific hypothesis.

[5] As *non sequitur* means an inference or conclusion that does not follow from established premises or evidence, we use sequitur to refer to a goal or action that follows from a previous goal or action.

[6] The borrowing of the logical terms antecedent and consequent should not be taken as an indication that the second goal logically follows from the first. The consequent only plausibly follows from the antecedent.

2.1 Case Suggestors

As noted above, a case provides an ordered set of actions that have worked in the past. Thus remembering a case appropriate to the current context suggests the next action. Multiple parts of multiple cases can be useful in solving a particular problem. Useful parts can be accessed directly, by retrieving the relevant part of a relevant case, or sequentially, by continuing to follow a previous case while it continues to be relevant. The two suggestors that use parts of previous cases are based on these two methods.

2.1.1 Sequential Access

If the results of running a step in the new situation match those obtained when it was run in the previous case, the next step in sequence in that case can be suggested. The **Continue-following-link** suggestor does this.

2.1.2 Direct Access

If the results are different, however, the **Case-snippet** suggestor uses direct access to part of a different case that can provide a suggestion of what to do next. Retrieval involves matching the current situation to the case part, or snippet's goal and context. In our system, CELIA, retrieval via direct access is first restricted to snippets that are centered around the goal type being considered. Then a weighted similarity metric is used, with matching occurring for all features within the context. The context includes the internal context, the results of actions taken up to that point in problem solving, so the retrieved piece is influenced by the results of goals pursued so far in this problem.

2.2 Causal Link Suggestors

Causal link suggestors use domain knowledge of function and structure to reason either forward or backward from a clause in the preceding goal instance in order to suggest the main clause for the consequent goal instance.

Variations in these heuristics include:

- Whether reasoning is forward or backward from the initial clause. For instance, reasoning backwards can lead toward suggesting hypotheses that could be root causes. Reasoning forward can lead to suggesting tests of hypotheses based on their potential effects.
- Which aspect of the previous goal instance to use as the initial clause. For instance, when reasoning from a test of a hypothesis a useful starting point is the test result. When reasoning from the interpretation of a test useful starting points include things ruled in or ruled out.
- Whether the initial clause is returned as a result when no progress is made in the causal chaining. When the consequent goal type is the same as the antecedent goal type, this is not appropriate.
- Whether to return a contradiction of the linked clause, or just the linked clause. For instance, a test for a contradiction of something that follows from a hypothesis can be a good test of the hypothesis.

2.3 Refinement Suggestors

Refinement suggestors use part/whole knowledge to suggest a new goal instance through refinement of the preceding goal instance. Either

- Its component is below the previous goal instance's component in the partonomy, (a *leak in the fuel line* is more refined than a *leak in the fuel system*). The previous goal instance's component is refined to a component that is part of the previous component. or
- The component is the same and the new predicate is more specific, (the *ECM not being grounded properly* is more refined than a *malfunction in the ECM*). This requires use of knowledge of the functions of the involved components. If the predicate is 'malfunction', then those predicates that are involved in obstacles to the component's function are considered as refinements of the previous predicate.

Variations in these heuristics include:

- Whether a clause that is equally as refined is acceptable. When the consequent goal type is the same as the antecedent goal type, this is not appropriate.
- Which aspect of the previous goal instance to use as the initial clause. For instance, refining the interpretation of a test, useful starting points include things ruled in or ruled out.

3 Controlling Suggestions: Restrictors

There are, in general, large numbers of possible next steps that could be generated by the methods above. Restrictors constrain the suggestion process so that effort is not expended trying to generate actions in directions that will not prove fruitful. In general, restrictors rule out goal sequences that are sometimes possible, but are not appropriate in the particular current circumstance. For instance, a test should not follow from a hypothesis that has already been tested, and a hypothesis should not follow from a hypothesis that has already been tested. However, a hypothesis can follow from a hypothesis that has already been refined, it could be another refinement. These examples suggest two of the restrictor heuristics.

No-sibs restricts a goal following from a previous goal to contexts in which no action has already followed from the antecedent.

Only-same-type-sibs restricts a goal following from a previous goal to contexts in which either no action has already followed from the antecedent, or contexts in which only actions fulfilling the same goal type have already followed from the antecedent.

Because some goal types should only follow from the most recent instance of some other goal types, restrictors are necessary for that purpose. For example, an interpretation of a test should follow from the most recent test instead of some previous test. This is clearly not the case for all goal sequences. For example, a number of hypotheses could be advanced, then tested in order, thus the test would not follow from the most recent hypothesis.

Most-recent restricts a goal following from another goal to contexts in which the previous goal was the most recent instance of that goal type.

Also needed are restrictors that constrain what can follow from the interpretation of a test. After a test result has been interpreted, what follows depends on the interpretation. If the hypothesis that is being pursued has been ruled in, either the hypothesis can be refined further, or a repair can be made. If nothing has been ruled in, and something ruled out, it is possible that the complaint should be further clarified. The following two restrictors are used.

Prev-ruled-out restricts a goal following from a previous action to contexts in which the previous actions included ruling out some condition.

Prev-ruled-in restricts a goal following from a previous goal to contexts in which the previous actions included ruling in some condition.

4 Selectors

Even with restriction, several steps might plausibly follow the current situation. Selector heuristics choose the best of those generated. Selectors work in two stages. Before suggestors are run, some selectors specify allocations of computational effort to the different suggestors associated with each of the possible future goal sequences. Then, after generation of plausible next steps, the best next step is chosen based on the rest of the selectors, and the amount of the allocation used.

The selectors that influence allocations include:

1. Allocate more effort to generating possibilities following from more recent goals pursued.
2. Allocate more effort to generating possibilities following from a leaf node of problem solving.
3. Allocate more effort to generating possibilities following from a problem solving node closer to the most recent goal pursued.

These allocations serve to limit the processing suggestors can do before cutting off search. This is important because it keeps the slowest heuristics, such as causal chaining, which can be intractable, from slowing the process down too much.[7]

When suggestions have been made, the choice of what goal and action to take is based on the percentage of allocated effort used in conjunction with the following preferences:

1. Favor possibilities generated using continue-following-link, then case-snippet, then other methods such as causal chaining.
2. Favor possibilities generated using goal sequences judged more likely in our analysis of the diagnostic task.
3. Favor possibilities generated from reasoning goals with more restrictor heuristics. These are less likely to have inadvertently escaped restriction.
4. Favor possibilities generated from reasoning goals with fewer suggestor heuristics. These are less likely to be low quality 'shots in the dark'.

The combined effect of the selectors is to favor continuing following along from the most recent goal, using a previously retrieved case snippet, or a newly retrieved case snippet. The preference is not absolute, however. It does not make the easiest suggestor heuristics dominant, because the allocated effort can vary widely based on the factors discussed above.

5 Example

We will illustrate the process of choosing the next action using the example shown in Figure 2. This is an English-ized version of a sequence of problem solving steps generated by our program CELIA.[8] The problem solver first clarifies the complaint, then verifies the complaint to make sure that the problem can be recreated. The problem solver hypothesizes that the carburetor is malfunctioning, then refines that guess. The idle speed is considered, and rejected. The idle mixture is considered, tested, and repaired, and yet the problems remain. A further hypothesis of the throttle dashpot being out of place is generated, and tested.

1. Clarify the complaint
2. Verify the complaint
3. Generate a hypothesis - carburetor malfunction
4. Generate a hypothesis - low idle speed
5. Test hypothesis - temperature of engine when stalling occurs (warm)
6. Interpret Test - idle speed not a problem; idle mixture possible problem
7. Repair - Adjust idle mixture screw
8. Test Repair - engine still stall? (yes)
9. Interpret Test - idle mixture not the problem
10. Generate a hypothesis - throttle dashpot out of place
11. Test hypothesis - distance between throttle dashpot stem and throttle lever small? (no)

Figure 2: Example Multi-step Solution in Automobile Troubleshooting.

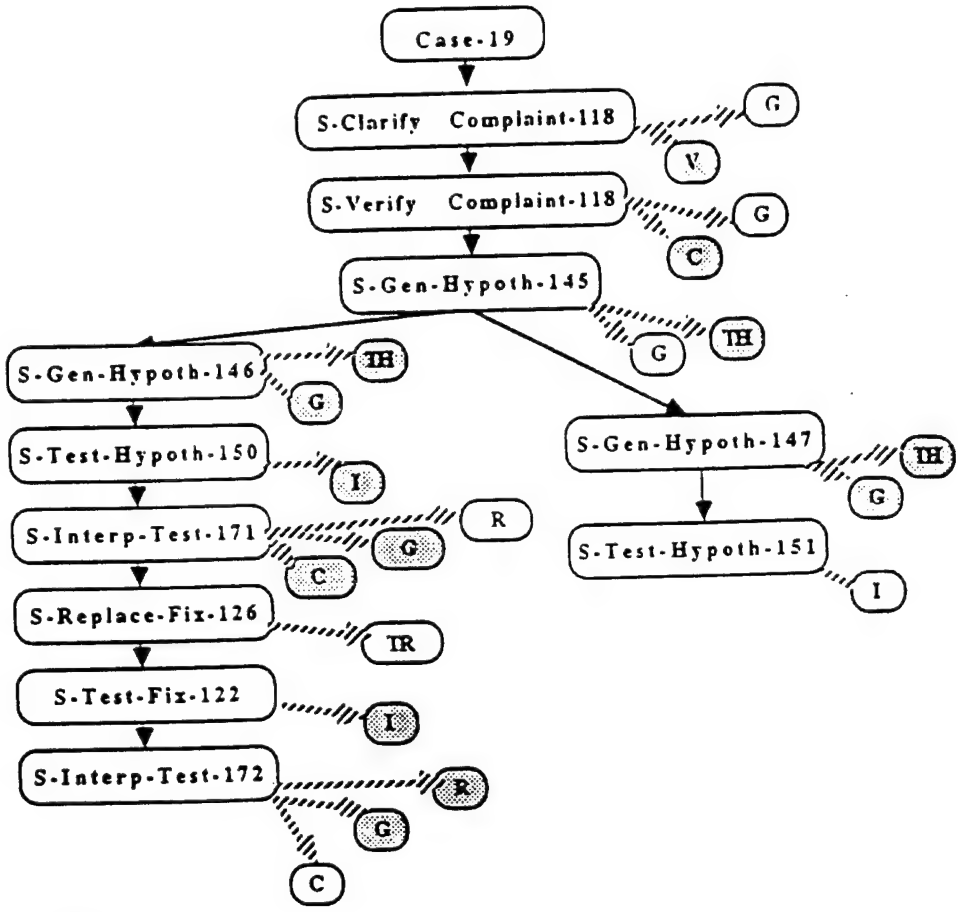
After step 11, the problem solving can be illustrated by the large nodes of the tree shown in Figure 3. Nodes represent goals that have been pursued so far. Links represent the sequencing relationships between the goals. S-Verify-Complaint-118 corresponds to the first step, S-Gen-Hypoth-145 corresponds to step 3 - Generating a hypothesis, in this instance a carburetor malfunction. The most recently completed action is included in S-Test-Hypoth-151. At this point the next action must be generated.

The small ovals in Figure 3 show types of possible succeeding goals that can follow from the parts of the problem solving to this point. The key for the different goal types is given.

The set of possibilities can be reduced significantly using the restrictor heuristics. Shaded ovals in Figure 3 show the effects of the restrictors. These are the directions restrictors have determined not to be fruitful.

[7] Causal chaining is constrained both by the strategy of trying to form a connection between actions rather than trying to form a connection over the large space between complaints and root causes, and by selection allocations.

[8] Actually, it was generated as predictions of what an expert would do by the learning component of CELIA using the same methods as described for the problem solving component. The problem solving component has not yet had the equivalent upgrade from the previous version.



The nodes represent the different goals that have been pursued so far. The links represent the sequitur relationships between the goals. The small ovals connected to nodes with striped arrows represent the possible goal types that can follow from the goal types of the nodes.

Figure 3: Remaining Possible Next Goal Types after restriction.

For each of the remaining possibilities there are several applicable suggestors, these are able to generate 26 possibilities including:

Goal Type	Instance	Following from Piece	Using Suggestor
G-Interp-Test	(Incorrect (Position Throttle-Dashpot))	S-Test-Hypothesis-151	Case-snippet
G-Gen-Hypoth	(High (Contains Carburetor-Float-Bowl Fuel))	S-Verify-Complaint-118	Case-snippet
G-Test-Repair	(Small (Dist Throttle-Dashpot-Stem Throttle-Lever))	S-Replace-Fix-126	Case-snippet
G-Test-Repair	(Low (Position Idle-Mixture-Screw))	S-Replace-Fix-126	Un-Improve
G-Test-Repair	(Increase (Position Idle-Mixture-Screw))	S-Replace-Fix-126	Equivalent
G-Replace-Fix	(Lean (Position Idle-Mixture-Screw))	S-Interpret-Test-171	Fault-Determination
G-Gen-Hypoth	(Hole Carburetor-Barrel)	S-Gen-Hypoth-145	More-Refined
G-Gen-Hypoth	(Clogged Carburetor-Pipe-To-Venturi)	S-Gen-Hypoth-145	More-Refined

From among these, the selector heuristics choose the first action, the interpretation of the test ruling out the hypothesis of the throttle dashpot being out of place. This suggestion was chosen due to several factors:

1. It was generated from the most recent previous goal and actions. Therefore, the suggestor which generated it was allocated a high amount of processing, of which not much was used in retrieving the case snippet that suggested the interpretation.
2. It was generated from a case snippet. Therefore it was favored at selection time.
3. Tests (of hypotheses or of repairs) need to be interpreted, so the judged likelihood of an interpretation of a test following from a test of a hypothesis is high, favoring this suggestion.

6 Related Work and Conclusions

A number of other efforts share some flavor with our approach. Koton [1988] combines use of a number of reasoning methods. First, associations formed from generalizations of cases are tried, then cases, and lastly model knowledge. However, the strict ordering of methods used is less flexible. More importantly, her approach does not generate steps for a multi-step solution, but rather a classification. Carbonell [1986] generates steps for a multi-step solution using a previous case. Domain knowledge is used in adapting the solution, but *one* case will either provide a whole solution or have to be abandoned or adapted. Parts of multiple cases cannot be used. Allen and Langley [1989] generate multi-step solutions using a combination of generalizations, cases, and domain knowledge (in the form of operators). However, they do not retain relations between problems and subproblems, so their DAEDALUS system cannot use an entire previous plan from memory.

Our approach combines the use of several types of knowledge and reasoning techniques. It takes advantage of knowledge about the relationships between goal types to provide constraint on the problem of coming up with the next action to do. The problem solving is flexible and can take advantage of the results of previous actions when deciding what to do next, while remaining goal directed. The approach has three phases – restrictors limit the number of possibilities to be considered, suggestors generate possible next actions, and selectors chose the action to take. There are several advantages to the approach. It combines multiple reasoning methods in a flexible manner. Problem solving is flexible enough to use whatever knowledge is available, using cases when appropriate cases can be found, domain knowledge when it can be useful. It is a flexible way of using parts of multiple cases in forming a solution that is a synthesis of steps. Problem solving can change directions when the results of the problem solving make that necessary. A major side benefit is that many of the suggestor heuristics can benefit when further knowledge is added to the system, in the form of new cases or new domain knowledge. Our system, CELIA, is a learning system, and is designed to acquire such knowledge. This makes problem solving more effective without having to learn new heuristics.

References

Allen, J. A. & Langley, P. (1989). Using concept hierarchies to organize plan knowledge. In *Proceedings of the Sixth Annual International Workshop on Machine Learning.*

Carbonell, J. (1986). Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In Michalski, R., Carbonell, J., & Mitchell, T., (Eds.). *Machine Learning: An Artificial Intelligence Approach, Volume II.* Morgan Kaufmann, Los Altos, CA.

Kolodner, J. & Simpson, R. J. (1984). A case for case-based reasoning. In *Proceedings of the Sixth Annual Conference of the Cognitive Science Society*, Hillsdale, NJ. Lawrence Erlbaum Associates.

Koton, P. (1988). *Using experience in learning and problem solving.* PhD thesis, Massachussetts Institute of Technology, Cambridge, MA.

Lancaster, J. & Kolodner, J. (1987). Problem solving in a natural task as a function of experience. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, Hillsdale, NJ. Lawrence Erlbaum Associates.

Lancaster, J. & Kolodner, J. (1988). Varieties of learning from problem solving experience. In *Proceedings of the Tenth Annual Conference of the Cognitive Science Society*, Hillsdale, NJ. Lawrence Erlbaum Associates.

Redmond, M. (1989a). Combining case-based reasoning, explanation-based learning and learning from instruction. In *Proceedings of the Sixth Annual International Workshop on Machine Learning*, San Mateo, CA. Morgan Kaufmann.

Redmond, M. (1989b). Combining explanation types for learning by understanding instructional examples. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, Hillsdale, NJ. Lawrence Erlbaum Associates.

Redmond, M. (1989c). Learning from others' experience: creating cases from examples. In *Proceedings of the Second Workshop on Case-Based Reasoning*, San Mateo, CA. Morgan Kaufmann.

Improving Case Retrieval Through Observing Expert Problem Solving

Michael Redmond*
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280
E-mail: redmond@cc.gatech.edu

Abstract

As case-based reasoners gain experience in a domain, they need to improve their case retrieval so that more useful cases are retrieved. One problem in doing this is that the reasoner who most needs to learn is least able to explain successes or failures. A second problem is that uncontrolled pursuit of an explanation could be very expensive. There are three keys to the approach presented. First, the student observes expert problem solving and sets up expectations for what the expert will do next. When expectations fail, the reasoner has its failure isolated to a single step, and the correct action for the situation has been provided. Second, if the student can retrieve part of a case that would have suggested a correct prediction, then that case snippet can be used to limit the explanation process, making the process more efficient. Third, when no explanation can be found, the reasoner resorts to empirical adjustment of feature importance.

Case-based reasoning (CBR) is based on the observation that experience, retained in the form of cases, can be used to efficiently and effectively solve future, similar problems. A case-based reasoner can improve in a number of different ways. It can acquire new cases. Or it can improve its case retrieval, so that more useful cases are retrieved. In Redmond (1989b) we discussed our approach to acquiring new cases through apprenticeship. Part of apprenticeship involves observing and understanding expert problem solving. This same kind of experience can be used to improve retrieval of cases.

Improving case retrieval is one of the key issues in case-based reasoning. Novice reasoners are frequently most influenced by surface features in retrieving previous experiences (Ratterman and Gentner 1987; Ross 1987). In becoming more expert, a reasoner must learn to retrieve *usefully* similar cases.

Much of the work on improving case retrieval has focused on learning indices though explaining a reasoner's own successful or unsuccessful problem solving. Learning indices based on explanations requires at least three things:

1. Realizing the need to learn.
2. Determining what the correct result should be.
3. Assigning credit for successes or blame for failures.

The problem is that these can be hard. A novice reasoner is the *most* in need of improvement and the *least* prepared to learn. A novice may not be able to generate the correct result. A novice may also have trouble assigning credit or blame. How can a novice get around these problems?

Apprenticeship can provide assistance with this problem. A novice can observe an expert solving a problem. A good student, who actively follows along with the example, sets up expectations at each point in the expert's problem solving. When the expectations are incorrect then he has a failure. The student thus has immediate feedback. The student realizes the need to learn. The student has been given the correct result. Most important, the failure has been isolated to a single step. This tighter feedback loop enables learning when it otherwise might not be possible, and makes learning more efficient.

An example will illustrate some of the issues involved in improving case retrieval through observing an expert. A student is observing an instructor solve an automobile diagnosis problem. This is part of an ongoing mentor relationship. The instructor has checked whether the car stalls when cold. The instructor has adjusted the idle mixture screw and determined that the car still stalls. The instructor has tightened any loose spark plugs, and determined that the car still stalls. He has checked if the throttle dashpot is out of place (it wasn't), and if the fuel level in the carburetor was too high (it was). Figure 1 shows the effect of all this on the problem solving context and also shows some of the more general features of the problem that were elicited by the instructor. At this point the reader need only note that there are a

*This research has been supported by the Army Research Institute for the Behavioral and Social Sciences under Contract No. MDA-903-86-C-173, and Contract No. MDA-903-90-K-0112 and by DARPA contract F49620-88-C-0058 monitored by AFOSR. The author wishes to thank Janet Kolodner for her advice and guidance, and Ashwin Ram for helpful comments on earlier versions of the paper.

Accepted for the 1991 Cognitive Science Conference, Chicago, Illinois

Current Context
Complaint: Car Stalls Out
Other Symptoms: None Reported
Frequency of Problem: intermittent
Temperature When Fail: Cold
Type of Car: 1980 Chrysler LeBaron
Car Owner: Julie Crider
Mechanics Involved: Tom Davis
Ruled In:
Spark Plug Connections Ok
Idle Mixture Ok
Carburetor Fuel Level High ***
Ruled Out:
Low Idle Speed
Lean Idle Mixture
Incorrect Position Throttle Dashpot
Tests Done:
Car Stalls When Cold?
Spark Plugs Loose?
Car Still Stalls?
Too Small a Distance Between Throttle Dashpot Stem and Throttle Lever?
Too High Level of Fuel in Carburetor Float Bowl?
Test Results:
Car Stalls When Warm
All Loose Spark Plug Connections Tightened
Car Still Stalls
Distance Between Throttle Dashpot Stem and Throttle Lever = 2cm
Distance Between Carburetor Float and Choke Chamber Surface = 3cm
Fixes Done:
Increase Position Idle Mixture Screw
All Loose Spark Plug Connections Tightened

Figure 1: Part of Current Context.

number of contextual features, including complex values for things that had been ruled in and ruled out, tests done etc.

At this point, the student has set up an expectation using a part of a case from his episodic memory. The student had previously been a participant or an observer of that step. Figure 2 shows most of the context at the

Incorrect Context
Complaint: Car Stalls Out
Other Symptoms: None Reported
Frequency of Problem: intermittent
Temperature When Fail: Cold
Type of Car: 1979 Chrysler Cordova
Car Owner: Bill Moss
Mechanics Involved: Tom Davis, Kevin Cousins
Ruled In:
Spark Plug Connections Ok
Idle Mixture Ok
Idle Speed Ok
Ruled Out:
Lean Idle Mixture
Low Idle Speed
Incorrect Position Throttle Dashpot
Idle System Leak Air
Tests Done:
Car Stalls When Cold?
Spark Plugs Loose?
Car Still Stalls?
Too Small a Distance Between Throttle Dashpot Stem and Throttle Lever?
Idle System Leak Air?
Test Results:
Car Stalls When Cold
All Loose Spark Plug Connections Tightened
Car Still Stalls
Distance Between Throttle Dashpot Stem and Throttle Lever = 2cm
No Apparent Air Leaks
Fixes Done:
Increase Position Idle Mixture Screw
Increase Position Idle Speed Screw
All Loose Spark Plug Connections Tightened

Figure 2: Portions of incorrect context.

Correct Context
Complaint: Car Stalls Out
Other Symptoms: None Reported
Frequency of Problem: daily
Temperature When Fail: Cold
Type of Car: 1981 Chrysler Cordova
Car Owner: Paul Crider
Mechanics Involved: Kevin Cousins
Ruled In:
Spark Plug Connections Ok
Idle Mixture Ok
Carburetor Fuel Level High ***
Idle Speed Ok
Ruled Out:
Lean Idle Mixture
Idle System Leak Air
Tests Done:
Car Stalls When Cold?
Spark Plugs Loose?
Car Still Stalls?
Idle System Leak Air?
Test Results:
Car Stalls When Cold
All Loose Spark Plug Connections Tightened
Car Still Stalls
No Apparent Air Leaks
Fixes Done:
Increase Position Idle Speed Screw
All Loose Spark Plug Connections Tightened

Figure 3: Portions of correct context.

time of the predicted action's previous occurrence. For short, we call this the *'incorrect' context.* The student's experience suggests the hypothesis that the choke linkage is sticking. We call this the *'incorrect' prediction* to indicate that it is not an appropriate prediction for the current situation. The instructor makes a different hypothesis, that the carburetor needle valve leaks. The student could have made this prediction, which we call the *'correct' prediction.* In the past he had observed the instructor taking that action. The context from that previous time is shown in Figure 3. We will call that the *'correct' context.* Why is the instructor's action a better choice for the current problem solving? The incorrect context matches a good number of the features of the current problem. The key difference favors the instructor's action, however. The information in the current context that the carburetor fuel level is high is the key difference. This information favors the correct hypothesis that the carburetor needle valve leaks. How can the student improve his case retrieval so that he will make the correct prediction in the future?

When the student's expectations are not met, then the student realizes the need to learn. Apprenticeship also takes care of the need for the student to know the correct action. The action taken by the instructor is assumed to be correct. The student still must determine what features of the current situation make the correct prediction appropriate, and/or make the incorrect prediction inappropriate. The student's exposure to that action in a previous example helps. He knows the context in which the action was previously taken. There are a number of similarities and differences among the contexts. A purely empirical approach will place some of the credit or blame on some spurious features that don't make a difference. On the other hand, if the student

tries to analyze all of the feature values in the contexts and explain why one prediction is more relevant than the other, that will be expensive. Also, we certainly can't assume that a student will always be able to explain why one prediction is more appropriate. How can these opposing forces be reconciled?

Our solution makes use of a combination of analytical and empirical methods. Similarity-based methods are used to focus explanation. When the student is able to explain the appropriateness or inappropriateness of a prediction, the associated case part is marked with that indication. The indication can be positive, that the prediction made by that part of the case is appropriate in a situation, in which case we call it an index. The indication can also be negative, that the prediction is not appropriate in a situation. We call the latter a *censor*. In addition, the student empirically adjusts the importance of matching different features. We first present the analytical approach to learning indices and censors. Then we discuss the empirical adjustment.

Analytical Approach - Learning Indices and Censors

The purpose of learning indices and censors is to improve prediction and diagnosis when they are carried out through CBR. Redmond (1990a) discussed our case representation. Briefly, cases are divided into *snippets*, each of which contain the information relevant to the pursuit of one primitive goal. Predictions are generated by retrieving a case snippet from memory. Each snippet contains the context in which it occurred. This enables similarity assessment, as well as the comparisons between correct and incorrect contexts mentioned above. An index is a particularly salient set of features of the context in which the snippet occurred. During retrieval, if a situation matches some of the indices, then that significantly increases the possibility that that snippet will be retrieved to provide guidance. The indices can be *parts* of contextual features. For example, *one* of the test results found in the problem solving leading up to the snippet might be marked as an index. A censor is a state (part of a feature) that suggests that a case snippet is not appropriate in that situation. If in a future situation that state exists, then the snippet can be rejected during retrieval.

We call our method ***analytical feature comparison***. It has five steps which involve distinguishing responsible features through first comparing feature values, then trying to explain the relevance of differences. The initial feature comparisons are similarity-based, the analysis comes into play in latter steps. The opportunity to apply the process occurs when the student, observing the instructor, uses a case snippet to incorrectly predict the instructor's action. The process starts out by retrieving a snippet that would have predicted the instructor's

Current Context Ruled In:
Spark Plug Connections Ok Idle Mixture Ok
Carburetor Fuel Level High
(a): Part of the current problem solving context.

Correct Snippet (case-generate-hypoth-305) Context
Ruled In:
Spark Plug Connections Ok +++ Idle Mixture Ok +++
Carburetor Fuel Level High +++ Idle Speed Ok
(b): Part of the correct snippet's problem solving context.

Incorrect Snippet (case-generate-hypoth-312) Context
Ruled In:
Spark Plug Connections Ok +++ Idle Mixture Ok +++
Idle Speed Ok
(c): Part of the incorrect snippet's problem solving context.

Figure 4: Portions of snippets involved in learning an index.

action. For short, we will call this snippet the *'correct' snippet*. The action itself is an effective additional cue that enables the correct snippet to be retrieved now. We will call the snippet that the student used to make the incorrect prediction the *'incorrect' snippet*. After the correct snippet has been retrieved, the process is ready to begin.

1. Eliminate from consideration features which are the same in both snippets' contexts. Features compared include aspects of the diagnosis, such as the tests done and their results, things ruled out, etc., as well as domain dependent features, such as car type. Given the contexts in Figures 2 and 3, the complaint, the other symptoms, and the ambient temperature when the failure occurs are the same in both snippets. Therefore, they don't provide a way of distinguishing why the correct snippet is appropriate in the current context. These are therefore no longer considered candidates for explanation.

2. Compare the remaining features with the current problem solving context. Features in which the current context better matches the correct context than the incorrect context are selected as candidates. It is more likely that something in those features would indicate that the correct snippet is appropriate. For example, in Figure 1 the current context's 'ruled in' feature had the value shown in Figure 4(a). The correct context and the incorrect context had the 'ruled in' values shown in Figure 4(b) and 4(c). Matches to the current context are shown by '+++'s. The incorrect context's 'ruled in' was not as close a match as the correct snippet's 'ruled in'. Thus 'ruled in' will continue to be included in the candidates. At the same time, the incorrect snippet's 'fixes done' is a better match to the current context than the correct snippet's 'fixes done'. Therefore, 'fixes done' will be eliminated from consideration for explanation. In the example, this step eliminates the frequency of the problem, the items 'ruled out', and the mechanics involved, since they favor retrieval of the incorrect piece.

```
Correct Snippet (case-generate-hypothesis-305) Context
  Type of Car:          1981 Chrysler Cordova
  Car Owner:            Paul Crider ***
  Ruled In:
      Spark Plug Connections Ok
      Idle Mixture Ok
      Idle Speed Ok
      Carburetor Fuel Level High ***
  Tests Done:
      Car Stalls When Cold?
      Spark Plugs Loose?
      Car Still Stalls?
      Idle System Leak Air?
      Too High Level of Fuel in Carburetor Float Bowl? ***
  Test Results:
      Car Stalls When Cold
      All Loose Spark Plug Connections Tightened
      Car Still Stalls
      No Apparent Air Leaks
      Distance Between Carburetor Float and
      Choke Chamber Surface = 2cm ***
```

Figure 5: Remaining portions of correct snippet after second step of analytical feature comparison.

3. Isolate the parts of the correct snippet's feature values that cause the remaining features to better match the current context. This is to narrow the responsibility to *parts* of features which favor the retrieval of the correct snippet in the current context. These parts of features are the best candidates for distinguishing why the correct snippet is appropriate in the current context. In the current example, for items 'ruled in', the isolated part is that the carburetor fuel level is high. This is the clause that makes the correct snippet's 'ruled in' better match the current 'ruled in'. Figure 5 shows the features remaining after the second step of the process. The parts of features isolated by this third step are marked with '***'s. This is where the explanation process will be focussed. This completes the initial feature comparison, which limits the analytical reasoning done.

4. Try to explain the significance of each of the remaining parts of features. The learner can try to relate each to the current action taken by the instructor. The student in the example tries to explain the relationship between the 'Ruled In' that the carburetor fuel level is high and the hypothesis that the carburetor needle valve leaks. An explanation that the student could (depending of his level of knowledge) construct is that a leaking carburetor needle valve could enable fuel to keep flowing into the carburetor float bowl, thus causing the fuel level to become high.

For learning censors a greater variety of relationships are useful. The learner can try to relate each remaining feature part to the current action taken by the instructor, or to the action suggested by the incorrect snippet. The relationships themselves can be different. A part of the current context relating to a 'contradiction' of the incorrect action is a good indication of a need for a censor. For example, if in a second problem *Carburetor Fuel Level Normal* has been 'ruled in', that contradicts an incorrect snippet's hypothesis that the carburetor needle valve leaks.

5. If such a relationship can be found, then the partial feature value is marked as an index or censor. In the main example, *Ruled In: Carburetor Fuel Level High* can be saved as an index to the correct snippet. In the same or similar situation in the future the snippet will then be more likely to be retrieved as a source of predictions or diagnostic actions.

In the second example, the incorrect snippet can be annotated with the censor *Ruled In: Carburetor Fuel Level Normal.* In the same or similar situation in the future the snippet will not be retrieved as a source of predictions or diagnostic actions.

We should note some limitations of our approach to learning indices and censors. First, the matching of parts of features has to be exact. Some form of knowledge-based matching like that used by Bareiss (1989) and Koton (1988) is needed. Fortunately, substituting knowledge-based pattern matching for simple matching doesn't affect the method. It just substitutes better reasoning capability in steps 1 through 3. Second, there is no attempt to learn indices that involve a conjunction of feature parts. There are certainly situations in which such an index would be necessary. With better explanatory capability, the student could learn more sophisticated indices. Some of the limitations on explanation may need to be relaxed though, costing the process some efficiency. Both of these limitations will be addressed in future work.

Empirical Feature Comparison

Sometimes a learner may not be able to explain why a prediction is wrong, or why another one might be better. The learner is not an expert. When he cannot, he needs to resort to less powerful, less knowledge-intensive methods. He can attempt an empirical approach to improving case retrieval. As with learning indices and censors, apprenticeship helps identify the need to learn. It also isolates the problem to a single step, and provides the correct action for the situation. As with learning indices and censors, the learner can attempt to retrieve part of another case that would have suggested a correct prediction. If such a case snippet can be retrieved, then the contexts can be compared. The differences can be used to empirically adjust the weights on features in the matching function. The adjustment method places greater importance on features that match and lead to correct predictions. It places less importance on features that match and lead to incorrect predictions. We should note that there are results that suggest that people can learn to distribute their attention among features, giving different weights to each (Nosofsky 1987). We originally discussed this approach in (Redmond 1989a). We have now integrated it with the analytical approach discussed in the previous section.

Feature	'Incorrect'	'Correct'	Change in Importance
CAR-TYPE	Partial	No Match	less important
CAR-OWNER	Match	No Match	less important
COMPLAINT	Match	Match	no change
HOW-LONG	Match	Partial	less important
RULED-OUT	Partial	Match	more important
TESTS-DONE	Partial	Match	more important
FIXES-DONE	Partial	Match	more important
CURRENT-HYP	Match	Match	no change
WHEN	Partial	Slight	less important

Figure 6: Example empirical saliency adjustment.

Figure 6 shows how the empirical change is done on an example incorrect prediction. Those features of the current context that match the context of the correct snippet more closely than the context of the incorrect snippet are made more important. Those features of the current context that match the context of the incorrect snippet more closely than the context of the correct piece are made less important.

The student can also make empirical adjustments when he is successful. When the student correctly predicts the instructor's action, the features of the current problem solving context that matched the features in the previous case are made slightly more important. In future problem solving these empirical adjustments lead to the features considered important by the instructor having more influence on the case pieces retrieved.

Related Work

We have addressed the issue of making case retrieval better. We use a form of apprenticeship to isolate a failure, and to obtain the correct solution step. Retrieval of other previous case snippets and comparison of features is used to focus explanation. Through explanation, indices and censors are learned.

Other approaches have been suggested for learning indices and censors. Hammond (1986), and Simpson (1985)'s approaches explained instances of their own successful problem solving. This requires a significantly more expert reasoner because the correct steps must be generated. Hammond and Simpson also emphasized creating indices to avoid failures. However, in trying to explain failures their approaches had to consider all features. Our approach limits the consideration of features to a smaller set. In addition, our approach avoids having to match the situation to an abstraction, as in Hammond's approach, an expensive proposition. Therefore, our approach saves explanatory effort.

In Barletta and Mark's (1988) Explanation-Based Indexing approach, the reasoner attempts to explain the relationships of features of the case to the actions taken. Those features for which an explanation can be formed are made indices of the case. Our approach takes more information into account in order to limit explanatory effort. Thus our approach is more efficient.

Bareiss (1989)'s PROTOS created both censors that specified when a case was not appropriate, and 'difference links' which specified the condition in which one solution should be avoided and another used. But the instructor did all the distinguishing of which features should be considered in explaining the failure. PROTOS doesn't include the initial consideration of an extra case that our approach does. So our approach offers the benefit of increased efficiency of learning through initial similarity-based comparison to other cases.

At a high level, our approach shares similarity with Lebowitz (1986)'s suggestion to use similarity based measures to focus explanation. However, we have applied an instantiation of that general idea to learning indices and censors.

Empirical adjustment of feature importance was also suggested by Aha (1989) and Salzberg (1988) in the context of purely empirical instance-based learning. The contribution here is its use in conjunction with analytical approaches, as a fallback strategy when the reasoner doesn't possess the knowledge necessary to distinguish why an action is appropriate or inappropriate.

Evaluation

Through the process of observing an expert, a reasoner can significantly improve its ability to predict the expert's actions, and thus its ability to diagnose. The improvement comes through acquisition of new cases, learning indices and censors, and through adjusting feature salience. Our system, CELIA, which is based on the model, has been evaluated through presentation of a sequence of 24 examples of expert problem solving. Twenty random orders of the examples were presented. The performance measure was the average accuracy of the system's predictions of the expert's actions. The improvement over the course of exposure to examples is seen in the data presented in Figure 7.

An ablation study showed that the by the end of the example sequences, the effect of the learning methods is starting to be seen. We lesioned the functions which learn indices and censors, and which adjust feature saliency. By the last 6 examples in the sequences, the average performance advantage for including these approaches was two percent. This difference is not that large, but for some problems the advantage was as high as 10 percent.

We expect that the effects will be larger with greater experience. To this experience level, not that many indices are learned. [1] Many of these are learned late in the sequences, leaving little time for them to be useful. The system doesn't have a strong domain model, so in many instances it cannot explain the importance of different feature values. With greater experience, this problem will gradually become less of a factor. In addition,

[1] An average of 1.2 are learned per observed example. An average example has 37 steps, which become 37 snippets. The range is from 18 steps to 79 steps.

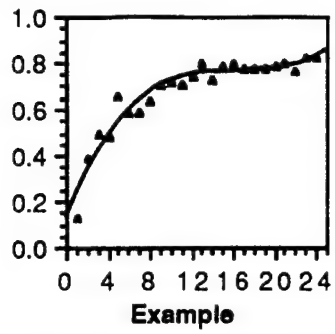


Figure 7: CELIA: Average results of exposure to sequence of examples. Average accuracy for reasoner predicting observed expert problem solving actions.

the student will have opportunities to use the indices and censors learned late in the test period.[2]

The approach has several advantages over previous approaches. The use of instruction through worked out examples, in conjunction with the student making predictions, shortens the feedback cycle, making learning more likely and more efficient. The retrieval and comparison of an additional case helps limit efforts at explanation. Third, the use of empirical adjustment of feature importance allows the student to become better at retrieving appropriate case pieces even when he is unable to explain failures.

Conclusions

In order for a case-based reasoner to improve its performance, it must acquire more cases and improve its case retrieval so that the right cases are retrieved at the the right times. When the reasoner uses part of a previous case to suggest an incorrect action, that case snippet needs to be marked as to what makes it not relevant in the current situation. If a part of a case would have suggested the correct action, that case snippet should be marked with the features of the current situation that make it relevant. The student must try to form an explanation based on the incomplete knowledge that he has.

Our approach to learning indices and censors offers the benefit of increased efficiency of learning through initial similarity based comparison to other cases and through immediate direct feedback available through observation of expert problem solving. Our case representation facilitates the learning process. Each snippet contains the pursuit of one primitive goal and the context in which it occurred. This enables the necessary comparisons between correct and incorrect contexts. The general approach allows the student to obtain the correct action for the current situation from the instructor, and attempts to distinguish when the correct and predicted actions are appropriate or not.

Learning indices, learning censors, and adjusting feature salience result in improving case retrieval. All three of these types of learning fit neatly within our general approach of learning by observing expert problem solving. Much of the power of CBR comes from retrieving useful cases, so this learning improves the case-based reasoner. Indices help the reasoner retrieve case snippets when they would be particularly relevant. Censors help the reasoner avoid being misguided by a snippet which may seem to be relevant but isn't. Improved knowledge of feature salience will lead to less consideration of superficial and spurious features during similarity measurement prior to use of indices and censors.

[2] It should be noted that some orders of presentation of examples will facilitate index learning. This ablation study did not take that into account. If example order can be controlled then learning can have a greater effect.

References

D. W. Aha. Incremental, instance-based learning of independent and graded concept descriptions. In *Proceedings of the Sixth Annual International Workshop on Machine Learning*, San Mateo, CA, 1989. Morgan Kaufmann.

R. Bareiss. Exemplar-based knowledge acquisition: a unified approach to concept representation, classification, and learning. Academic Press, New York, NY, 1989.

R. Barletta and W. Mark. Explanation-based indexing of cases. In *Proceedings of a Workshop on Case-Based Reasoning*, San Mateo, CA, 1988. Morgan Kaufmann.

K. J. Hammond. Learning to anticipate and avoid planning problems through the explanation of failures. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-86)*, San Mateo, CA, 1986. Morgan Kaufmann.

P. Koton. *Using experience in learning and problem solving.* PhD thesis, Massachussetts Institute of Technology, Cambridge, MA, 1988.

M. Lebowitz. Concept learning in a rich input domain: Generalization-based memory. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach, Volume II.* Morgan Kaufmann, Los Altos, CA, 1986.

R. M. Nosofsky. Attention and learning processes in the identification and categorization of integral stimuli. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 13:87–108, 1987.

M. J. Ratterman and D. Gentner. Analogy and similarity: Determinants of accessibility and inferential soundness. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, Hillsdale, NJ, 1987. Lawrence Erlbaum Associates.

M. Redmond. Combining explanation types for learning by understanding instructional examples. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, Hillsdale, NJ, 1989. Lawrence Erlbaum Associates.

M. Redmond. Learning from others' experience: creating cases from examples. In *Proceedings of the Second Workshop on Case-Based Reasoning*, San Mateo, CA, 1989. Morgan Kaufmann.

M. Redmond. Distributed cases for case-based reasoning; facilitating use of multiple cases. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-90)*, Boston, MA, 1990. Morgan Kaufmann.

B. H. Ross. This is like that: The use of earlier problems and the separation of similarity effects. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 13:371–416, 1987.

S. Salzberg. Exemplar-based learning: Theory and implementation. Technical Report TR-10-88, Harvard University, Center for Research in Computing Technology, 1988.

R. L. Jr. Simpson. *A Computer Model of Case-Based Reasoning in Problem Solving.* PhD thesis, Georgia Institute of Technology, Atlanta, GA, 1985.

Appendix F
CORA
Selected Papers

Martin, J. (1988). Retrieving Reasonable Predictions from Case Bases. In *Proceedings of the 1988 AAAI Case-Based Reasoning Spring Symposium.*

Martin, J. (1989). Focusing Attention for Observational Learning: The Importance of Context. In *Proceedings of IJCAI-89.*

From the Proceedings of the 1988 AAAI CBR Workshop.

Retrieving Reasonable Predictions From Case Bases[1]

Joel Martin
Georgia Institute of Technology
Atlanta, Georgia 30332-0260
E-mail:joel@gatech.edu (404)-894-5550

Abstract

Past experience is a major source for improving predictions in a reasoning system. These predictions, however, cannot be retrieved haphazardly. They must be retrieved when they are most likely to be correct and useful. Retrieving a prediction based on overall similarity between a past case and a current one is useful when nothing is known about a domain, but it is not enough in general. Even ranking different features by their importance is insufficient, because it does not take the retrieval goals or the context into account. A retrieval method is proposed that uses a form of context dependent importance ranking of features. This method actually learns the importance ranking appropriate for different goals and contexts. The learning provides a justification for the predictions made, and provides flexibility in changing domains. Further characteristics of the method for complex, abstract features are presented. Finally, possible future extensions are discussed.

1 Introduction

Any AI system that is to autonomously improve its performance must be able to learn from its experience, that is, from instances or cases. Presumably, information from certain past cases will somehow help predict portions of future cases or experiences. If, for example, Sarah became sick the last time she ate scallops, she may wish to avoid them in the future predicting that they again will make her sick. This example prediction seems reasonable but there are conceivable predictions that are not reasonable. Sarah, for instance, would not conclude that orange juice would make her sick just because scallops once made her sick. The first prediction is reasonable because there is some relation between the past and current cases. Reasonable predictions do not come from simple, haphazard use of past experience. Therefore, to make reasonable predictions from experience, an AI system must specify what characterizes good predictions and use those characteristics for retrieval.

A naive method for reasonable retrieval would search for the overall most similar past case and project some of its information to the current situation. The overall most similar past case is one that shares the most features with the current case. This is certainly a reasonable approach when little is known about the domain (Russell, 1987). When something is known, however, overall similarity can generate unreasonable predictions (Kolodner, 1989; Russell, 1987). For example, although $453 + 789 = 1242$ is very similar to $453 \div 789 =?$, neither the answer nor the reasoning method can be correctly projected to the new case. A more appropriate retrieval method would find the most similar match by primarily considering the *important* features of the present and past cases, such as the arithmetic operator in the above example. Actually, even this is not enough, because the *important* features can change depending upon the *retrieval goal.* For instance, when trying to predict whether Frank has

[1]The author wishes to thank Janet Kolodner and Mike Redmond for helpful comments on this manuscript. This research was supported by the Army Research Institute under Contract No. MDA-903-86-C-173.

an accent, his first language and birthplace are important aspects; whereas when trying to guess his age, hair color and wrinkles become important. The importance ranking of features then, must be dependent upon the *retrieval goal*[2] (Kolodner, 1989; Russell, 1987; Seifert, 1988). There is one additional requirement for reasonable retrieval. Even when the retrieval goal is the same, the *current context* can influence feature importance. For example, when someone buys books as gifts, features of the intended recipient can help determine which other features are important. When choosing a book for a child, the reading level or difficulty is very important, while the binding and copyright date are less important. However, when choosing a book for a book collector, binding and date become very important, while reading level is less so. To summarize the above, reasonable retrieval requires identifying similarities between the past and current cases. However, these similarity measures should only consider features that are, in some sense, important. Feature importance is dependent on both the retrieval goal and the retrieval context.

In order to actually retrieve reasonable predictions using importance, there must be some justification or rationale that defines which features are most important for prediction in any given situation. There are possibly many such rationales, but one that may be particularly useful is a probabilistic justification. In this view, feature importance depends on the conditional probability between values of the retrieval goal and the current context. The use of probabilities potentially allows identification of the most probable predictions. This view also allows learning of feature importance to reduce the burden on a programmer and to allow flexibility for domains in which feature importance varies over time. This paper argues for a retrieval method, embodied in a system called CORA-L, that learns context dependent importance values from cases.

2 CORA-L

Briefly, CORA-L maintains conditional probabilities between all pairs of feature values; and if those conditional probabilities are found to give poor predictions in some contexts, new features that are conjunctions of values are formed (Martin, 1988). This is essentially a distributed memory representation, not unlike some PDP approaches (McClelland & Rumelhart, 1986). Input cases are stored distributed across many features; and, if the given features are inadequate to predict aspects of the cases, then combinations of features are learned. Although this depiction of CORA-L is an oversimplification that ignores the complexity of realistic domains, it is argued that this simple notion is all that is needed to describe the retrieval of reasonable predictions from past cases. The next section discusses how CORA-L deals with more complex, realistic representations.

In CORA-L, predictions are retrieved from memory by using the known values to find the most probable unobserved value of a particular attribute. Specifically, stored conditional probabilities between values of the target attribute and the context are compared and the most likely prediction is made. For instance, suppose that a doctor sees a patient with a severe fever and a hacking cough, and that she wants to hypothesize what the underlying disease

[2]It is important to distinguish the retrieval or prediction goal from possible problem solving goals. The retrieval goal is essentially a specification on the type of value to predict. For example, a retrieval goal might direct retrieval to predict the value of a particular attribute. The problem solving goal, on the other hand, may be a goal that the retrieval goal is serving. CORA-L treats problem solving goals and constraints as any other attribute in the givens. The problem solving goal is not a priori given special status, but, in general, the goal will be highly informative and therefore more important.

might be. CORA-L would combine $P(strepthroat \mid cough)$ and $P(strepthroat \mid fever)$ by the following formula:

$$P(strepthroat \mid cough\&fever) = \frac{P(strepthroat \mid cough) \cdot P(strepthroat \mid fever)}{2}$$

This formula uses a version of an arithmetic average to estimate the probability with multiple givens (Martin, 1988). In general, the formula is,

$$P(B \mid A_1\&A_2\&A_3, ...) = \frac{\Sigma P(B \mid A_i)}{n}$$

For the above example, CORA-L would use this formula to perform similar calculations for alternative diseases, and the one with the highest probability would be predicted.

When the conditional probabilities with only one given feature, such as $P(strepthroat \mid fever)$, do not allow the generation of good predictions by the above formula, then new features are created by combining two previous features. In the above example, if 'fever' alters the importance or predictivity of 'cough' or vice versa, then a new conditional probability would be formed to explicitly keep track of $P(strepthroat \mid cough\&fever)$.

CORA-L's method has all the characteristics of reasonable retrieval outlined in the introduction. The use of similarity is weighted by feature importance and this weighting depends both on the present context and on the value or values to be predicted. Importance is implicitly defined as a kind of informativeness. Informativeness is basically the ability of a particular value to distinguish between the multiple possible values of the target attribute. A very informative value will provide strong support for one or two of the values of that attribute while providing very weak support for other possible values. For example, a yellowing of the whites of the eyes is very informative of hepatitis. In this scheme, those values that are most informative and hence most important with respect to the retrieval goal automatically have a greater effect on which value will be predicted than would less informative values. This definition is very similar to Swaminathan's (1988) proposal of choosing indices that increase associativity and discriminability, except that in CORA-L, these ideas are defined in terms of conditional probabilities. Sensitivity to the retrieval goal is possible, because multiple conditional probabilities and hence importance values are stored. That is, when the goal is to find a value for attribute-A, different conditional probabilities are used than when the goal involves attribute-B. Finally, because of CORA-L's learning of compound features, the present context can alter the informativeness of a given value, even if the retrieval goal remains the same.

CORA-L, therefore uses feature importance to retrieve predictions. As well, the importance ratings are dependent on both the retrieval goal and the context of the present case. Not only does CORA-L meets these necessary conditions for reasonable predictions, but it also uses empirically justified importance values. The law of large numbers implies that as the size of the case base increases so will the accuracy of CORA-L's conditional probabilities. The prediction accuracy will likewise increase.

3 Abstract Features

Most real-world domains have more complex structures than was implied by the simplifying assumptions in the last section. The JULIA (Hinrichs, 1988) domain of catering, for example,

uses multiple hierarchies to describe food items and requires ways to describe combinations of food items. The texture of chocolate liqueurs might be described as some type of combination of 'soft-chewy' and 'thin-liquid'. This example assumes combining relationships, like 'inside', and hierarchies of solid and liquid textures. To capture these more complicated structures, CORA-L allows hierarchies of features in which each parent describes a set of mutually exclusive alternatives (XOR), a set of alternatives (OR), or a combination of values (AND)[1]. The AND type of hierarchy corresponds roughly to the CORA-L's learned combinations as described above, but it also includes information about the relationship between the ANDed features. The OR and XOR hierarchies have probabilities associated with the children to allow for fuzzy or probabilistic concept definition. The use of these hierarchies greatly complicates the updating of probabilities and retrieval of predictions described above. However, it allows the use of the most informative features, at any level of abstraction, when retrieving predictions. Also, it allows the prediction of abstract features when more specific predictions are not well supported.

4 Concluding Remarks

A method is proposed for the retrieval of predictions and is shown to find reasonable and empirically justified predictions. Additional characteristics of the method for complex representations are presented. The complete paper will expand each of the above sections and in particular will present a running example to illustrate the method and its extension. The concluding section will suggest future directions such as learning about continuous numeric values and learning temporal sequences.

References

Hinrichs, T. (1988). Towards an architecture for open world problem solving. In *Proceedings of the DARPA Case-Based Reasoning Workshop.*

Kolodner, J. (in press). The mediator: analysis of an early case-based problem solver. *Cognitive Science, Cognitive Science.*

Martin, J. D. (1988). Cora: a best match memory for case storage and retrieval. In *Proceedings of AAAI Case-Based Reasoning Workshop.*

McClelland, J. L. & Rumelhart, D. E. (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol I.* Cambridge, MA: MIT Press.

Russell, S. J. (1987). *Analogical and Inductive Reasoning.* PhD thesis, Stanford University, Stanford, CA.

Seifert, C. M. (1988). A retrieval model for case-based memory. In *Proceedings of AAAI Case-Based Reasoning Workshop.*

Swaminathan, K. (1988). Properties of an indexing scheme. In *Proceedings of AAAI Case-Based Reasoning Workshop.*

[1]These types are related to the notion of partonomies (AND) and taxonomies (XOR, OR) but are more general and possibly more flexible.

accepted at IJCAI-89

Focusing Attention for Observational Learning: The Importance of Context

Joel Martin
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332-0260
E-mail:joel@gatech.edu

Area: Fundamental Problems, Methods, Approaches
Subarea: Cognitive Modeling

Abstract

When learning language or natural category structure, humans do not need instruction or explicit feedback; they learn through active observation. Computational models of this type of human learning not only aid in the description of human learning but also suggest possibilities for machine learning. A significant component of human observational learning is the ability to focus attention toward important or relevant input features. An attention mechanism with this capability can serve as an inductive bias to facilitate learning. Past attempts to model attentional focus for human learning have postulated a single salience value for each feature. Features with greater salience command more attention. However, these models assume that the feature's salience is not dependent on context, whereas studies of human attention show sensitivity to context. As well, sensitivity to context can allow more complex learning. This paper presents a mechanism for contextually focused attention in observational learning. The new mechanism is compared to the salience method. Discussion of the results includes predictions for human behavior and implications for machine learning.

Observational learning is a form of inductive knowledge acquisition in which there is no external guidance, such as explicit feedback. However, some guidance or learning bias is required to make general induction tractable (eg. VanLehn, 1987). Since humans do engage in some observational learning (Billman, Heit, & Dorfman, 1987), there must be a method for *internally* guiding this learning. Discovering what this method is will prove useful both for understanding human learning and for designing computer programs that learn from observation. Zeaman and House (1963) and Billman and Heit (1988) have argued that attention directed by learnable feature saliences may provide some

[1]The author wishes to thank Dorrit Billman for providing constructive comments on all aspects of this research, and Janet Kolodner for assistance with earlier versions of this paper. This research was supported by the Army Research Institute for the Behavioral and Social Sciences under Contract No. MDA-903-86-C-173.

internal guidance for human learners. They each proposed a mechanism for doing this and were able to confirm the approach for simple learning. Neither method used context, or what is already known about an example, to help focus attention. Other researchers, however, have found that human attention and other cognitive processes vary with context (Loftus & Mackworth, 1978; Barsalou & Medin, 1986). As well, the use of context can allow learning of more complex examples. The non context approach assumes that there is only one important subset of features that are always salient. As will be demonstrated below, when this assumption is violated, learning is not facilitated.

Given that attention is useful for human and machine observational learning, it is important to ensure that proposed attentional mechanisms support a useful type of learning. Human observational learning is most clearly useful for natural language and concept acquisition. Additionally, many machine learning studies of observational learning have concentrated on concept or category acquisition (Schlimmer, 1986; Fisher, 1987). These types of knowledge have frequently been described as capturing correlational feature structure (Rosch, 1978; Medin & Schaffer, 1983; Fisher, 1987). In other words, category structure and linguistic structure can be represented partially by correlational rules or conditional probabilities of the form: $P(feature1 = value1 \mid feature2 = value2)$. Thus, a "rule" such as, $P(covering = feathers \mid locomotion = wings)$, records the frequency with which 'feathers' occur given that 'wings' is true. Anderson (1988) has further argued that even if human category structure is not implemented using conditional probabilities, it and other phenomena are best described and explained by probabilities. Similarly, recent machine learning models of concept acquisition have proposed that categories can be best learned by maintaining conditional probabilities (Schlimmer, 1986; Fisher, 1987). These psychological and machine learning studies suggest that an adequate model of human and machine attentional learning should demonstrate how the attention mechanism can facilitate learning of conditional probabilities or estimates thereof.

This paper presents a new model of the use of attention for observational learning. This model, called Contextually Focused Sampling, introduces a context controlled attention mechanism, and is proposed as a method both for human observational learning and for machine learning. This use of context was partially motivated by the need for dynamic learning biases (eg. Rendell et al., 1987) and machine learning studies of the use of probabilistic context for generalization (eg. Fisher, 1987). CFS is compared to an important non-context alternative, Focused Sampling (Billman & Heit, 1988), to demonstrate its similar behavior for simple learning and its superior behavior for more complex learning in which there are multiple important subsets of features.

Focused Sampling

Billman and Heit's (1988) CARI implementation of the Focused Sampling (FS) method describes how attention can be used to facilitate observational learning. Put simply, FS allocates more attention to those features that participate in strong correlations or rules. The 'rules' and 'correlations' referred to by Billman and Heit are simply the conditional probability relationships between features.

In CARI, two features, such as color and size, are sampled, and a prediction of the value of the second feature is made on the basis of the value for the first feature. All training examples were assumed to be collections of feature/value pairs, and sampling a feature reveals that feature's value. For example, the color feature, when sampled, might be found to have the value 'green'. If the prediction of the second value is correct, then the saliences of the features and the strength of the prediction are incremented. Otherwise, these values are decremented. The adjustment of the values is based on an estimator of conditional probabilities called the delta rule. This estimator, in similar forms, has been used in many psychological learning models (Rescorla, 1972; Rumelhart, Hinton, & Williams, 1986). CARI updates the rule strength and feature saliences by,

$$S_n = S_{n-1} + \alpha[T - S_{n-1}]$$

S_n = Strength or Salience; α = learning rate parameter
$T = 1$, if prediction is correct; $T = 0$, otherwise.

FS is an attentional learning mechanism that supports learning of correlational structure (Billman et al., 1987; Billman & Heit, 1988); and there are two major learning behaviors of the model that any viable alternative must also demonstrate.

- First, FS produces a facilitation in learning as compared to random sampling of features (Billman & Heit, 1988);
- Second, particular rules are learned faster when they are part of a system of interrelated rules than when they occur in isolation. Billman and her colleagues term this effect *clustered feature facilitation.* Human subjects have demonstrated this effect for observational learning of a novel language (Billman et al., 1987).

Contextually Focused Sampling

Humans are able to use information that they already know about an example to direct their attention to unusual aspects of the same example (Loftus &

Mackworth, 1978). Secondly, some multiple-look attention models (Trabasso & Bower, 1968) suggest an averaging method for taking what is known into account for response generation. A final reason to suspect that context can be important for focusing attention in learning is that algorithms like FS would not allow a human or machine learner to focus on different cohesive subparts of an example. For instance, there are many subsets of animal features that internally cohere, like food-type and size or habitat and means-of-locomotion. FS, however, assumes that there is only one important subset. An alternative model will be proposed that introduces a limited form of context for feature sampling. The use of the word 'context' in this work refers specifically to known feature values of a particular example. Using context for attention therefore refers to using those feature values that have already been observed to help choose other features to which to attend.

The method proposed by this paper, Contextually Focused Sampling (CFS), based on their estimated predictability. It calculates those estimates based directly upon estimates of conditional probability between feature values. In CFS (Figure 1) then, choosing a feature depends upon that feature's predictability given what value are already known. This method allows the probability of sampling a particular feature to vary with the context.

- Choose a starting feature F_1 . The probability of sampling a feature is that feature's no-context salience divided by the sum of the no-context saliences for all features.
- Sample the value, v_1, for F_1 .
- Loop for $i = 2 .. n$ (where n is a parameter)
  - Choose feature F_i . Features are chosen that are best predicted by what is already known about an example ($v_1 \ldots v_{i-1}$). An estimate is made of each feature's predictability, and this is used to probabilistically choose a feature.
  - Find the value, v_i , for F_i in the example.
  - Increment the estimate of conditional probability for all of the form: $P(v_j \mid v_k)$, where j and $k = 1 .. i, j \neq k$.
  - When i=2, increment the no-context saliences as described by Billman & Heit (1988).

Figure 1: Contextually Focused Sampling Algorithm

The CFS algorithm, like the FS, uses the delta rule to update the estimates of conditional probabilities and no-context feature saliences. The no-context

feature saliences are used for sampling when nothing is yet known about an example. Feature saliences *in* context, because they are based solely on the estimates of the conditional probabilities, are indirectly updated. An important difference between CFS and FS is that CFS allows multiple samples to be taken from each example in order to provide context.

CFS, unlike FS, requires an algorithm for estimating predictiveness in context, i.e., when several features have already been sampled. It is not reasonable to maintain all such higher order probabilities, because there are exponentially many of them. The most straightforward alternative is to use a Bayesian estimate assuming independence (Martin, 1988). However, pilot studies have shown that an arithmetic average (Trabasso & Bower, 1968) is better correlated to actual higher order conditional probabilities for the types of training example being used. Davis (1985) gives an argument for the use of a geometric average in a similar machine learning system.

Experimental Tests of CFS

CFS was compared to FS in three experiments. The first two experiments were performed to demonstrate that CFS is a viable alternative to FS. Experiment III was conducted to determine whether CFS is superior to FS for more complex inputs.

Algorithms. The experiments performed comparisons between three algorithms, Random Sampling (RS), FS, and CFS. In general, it is difficult to compare algorithms because they often differ in more than one characteristic. For instance, CFS and FS differ not only by how a feature is sampled but also by how many features are processed per example. FS samples exactly two features, while CFS can sample several. These extraneous differences can confound a comparison on the characteristic of interest. It is therefore important to remove as many extraneous differences as possible before comparisons are made. The FS algorithm was modified to incorporate the loop from the CFS algorithm. The only difference between the FS and CFS algorithms was that the former always used salience to select features for sampling. The RS algorithm was like the CFS algorithm, except that it selected features independently of salience and estimates of conditional probabilities. These versions of the RS, FS, and CFS algorithms were used in all experiments.

General Method. The general method used for all three experiments was very similar to that used by Billman and Heit (1988). The input was provided as lists of digits in the form, (1 2 3 2), to represent that the features 1 through 4 have the values 1, 2, 3, 2 respectively. These number vectors are used for simplicity but are meant to represent vectors such as, (covering=fur, habitat=land,

size=big, locomotion=legs). In all three experiments, the inputs consisted of eight features (Figure 2). These inputs were presented one at a time as examples. Each trial consisted of presenting one example that was randomly selected from all available inputs. These trials were divided into blocks of 50. As in Billman and Heit (1988), the strengths between the values of the first two features were averaged to measure learning after each block of trials. In all sets of training examples, the first two features were strongly related. The test strengths were, $\{f_1 = 1 \rightarrow f_2 = 1, f_2 = 1 \rightarrow f_1 = 1, f_1 = 2 \rightarrow f_2 = 2, f_2 = 2 \rightarrow f_1 = 2\}$. Statistical comparisons were made based on the average target strength after a criterion number of trial blocks. The criterion was set for each experiment when the mean strength for RS was equal to 0.50 ± 2, as in Billman and Heit (1988).

a				b			
(1 1 1 1 1 2 1 1)	(1 1 1 1 2 1 2 1)			(1 1 1 2 1 1 1 1)	(2 2 1 1 1 2 2 1)		
(2 2 2 2 1 1 2 1)	(2 2 2 2 2 2 1 1)			(1 1 1 2 2 2 2 2)	(2 2 1 1 2 1 1 2)		
(1 1 1 1 1 2 2 2)	(1 1 1 1 2 1 1 2)			(1 1 2 1 1 1 2 2)	(2 2 2 2 1 2 1 2)		
(2 2 2 2 1 1 1 2)	(2 2 2 2 2 2 2 2)			(1 1 2 1 2 2 1 1)	(2 2 2 2 2 1 2 1)		

Figure 2: Input Vectors used in the Experiments.

For all experiments, the variable parameters were set to the values used by Billman & Heit (1988) and were held constant for all experiments. The initial strength values were set to 0.01, initial feature saliences were set to 0.125, and delta learning rates were set to 0.02. CFS and the modified FS and RS algorithms have one additional parameter, the number of features sampled per example. Pilot studies demonstrated that as this parameter raises, learning facilitation increases. This parameter was set at 3 samples per example for all algorithms throughout the experiments to reflect the limited capacity of attention and to achieve some benefit of context for CFS.

Fifteen simulated subjects were run in each condition. These subjects varied due to probabilistic sampling and random example selection.

Experiment I & II

CFS should be able to demonstrate the significant behaviors of Focused Sampling. The first of the two important FS behaviors is a facilitation of learning as compared to random sampling. In Experiment I, CFS was predicted to produce a learning facilitation because, like FS, CFS's focusing mechanism leads it away from irrelevant features.

CFS should also show increasing facilitation for greater numbers of interrelated features. As noted above, FS predicts facilitated learning for greater

numbers of clustered features. In Experiment II, CFS is predicted to show clustered feature facilitation because the CFS sampling is biased toward features that are interpredictive.

Method. In experiment I, the subjects received the inputs presented in Figure 2a. These inputs were chosen to maximize FS benefit by interrelating half the features (Billman & Heit, 1988, experiment 3). The remaining four features were termed irrelevant because they are each randomly related to any other feature. Experiment I compared between the different attention methods, random sampling, FS, and CFS.

The method for Experiment II was the same as for Experiment I, except that the both sets of inputs from Figure 2a and Figure 2b were used. Fifteen simulated CFS subjects received inputs with two clustered features and 15 received inputs with four clustered features. The learning measure used for each set of inputs was the difference in learning between using the CFS and RS algorithms.

Results. The learning rates depicted in Figure 3 show clear effects of attention method.

An ANOVA was performed using the strength values at the criterion number of trials as defined above. Attention method showed a significant effect, $F(2,42) = 33.66, p < 0.01$. Tukey's HSD was used to compare means for the attention method to determine significant differences: $HSD(3,42) = 0.079, p < 0.01$. The comparisons revealed that both CFS and FS showed facilitation over the random method. Although CFS produced a greater facilitation than FS, this difference was not significant. These data demonstrate that CFS produces the same type of learning facilitation as FS.

In Experiment II, there was an greater facilitation for clusters of four features rather than two. A t-test was performed at the learning criterion for the four clustered feature condition, $t(28) = 2.73, p < 0.01$. The test showed that, as with FS and in accord with human data, CFS demonstrated clustered feature facilitation.

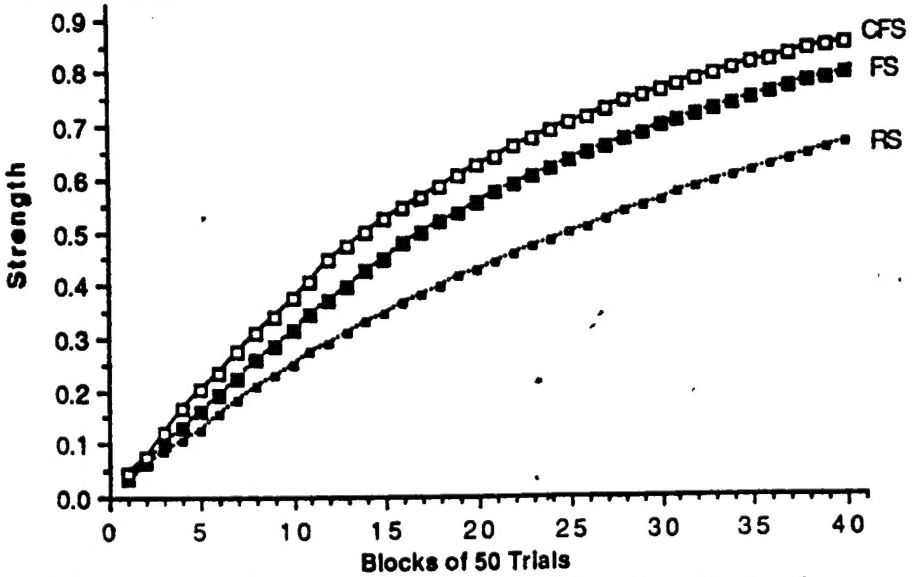


Figure 3: Learning with Different Attention Methods

Experiment III

As predicted, CFS produces two findings which motivated the FS model. It was assumed that because it was sensitive to context, CFS would predict greater learning facilitation for more complex inputs than would FS. Both Experiment I and II and the experiments of Billman and Heit (1988) have used inputs in which there was only one important cluster of features. That is, there are some relevant features and some irrelevant features, and all relevant features are intercorrelated. However, more realistic inputs would allow for multiple clusters of relevant features. For example, in humans, hair-color and eye-color are somewhat intercorrelated as are arm-length and height. All four of these features are relevant to feature clusters, but are not *all* intercorrelated.

Context is important for attentional learning in domains with multiple clusters because it allows the human or machine learner to concentrate on a single subcluster at a time. The non-context approach used by FS would set the saliences of the features independently of the cluster, permitting sampling across clusters. For example, if hair-color was the most salient and height the second most salient then the most frequent sampling pair would be across clusters. CFS can help alleviate this problem, because it allows the feature saliences to vary depending on what has already been sampled. In the above example, after hair-color was sampled, then the most salient feature became eye-color. The saliences in CFS are modified to have the learning focus on one cluster at a time.

Because of these considerations, it was predicted that CFS would be found to be superior to FS when there were multiple unrelated clusters of features in the input. As well, FS was expected to have a decreased learning facilitation as compared to random sampling.

Method. The method was the same as for Experiment I and II, except that the inputs had three clusters of three features each and three irrelevant features. The three clusters of features were independent of each other. All three algorithms were compared on these inputs. The learning measure, as in Experiment I, was the average strength of the rules relating features one and two; and statistical comparisons were made at the criterion number of trials.

Results. Figure 4 shows the learning curves for each block of 100 trials. There was an increased facilitation for CFS over FS and RS. An ANOVA was performed that indicated a significant difference between algorithms: $F(2, 42) = 31.21, p < 0.01$. Tukey's HSD, $HSD(3, 42) = 0.082, p < 0.01$, revealed a significant difference between CFS and both other groups. FS was not found to be significantly different from random sampling.

As predicted, CFS was superior to FS and RS for inputs with multiple clusters. As well, the multiple clusters prevented a significant learning facilitation for FS over RS.

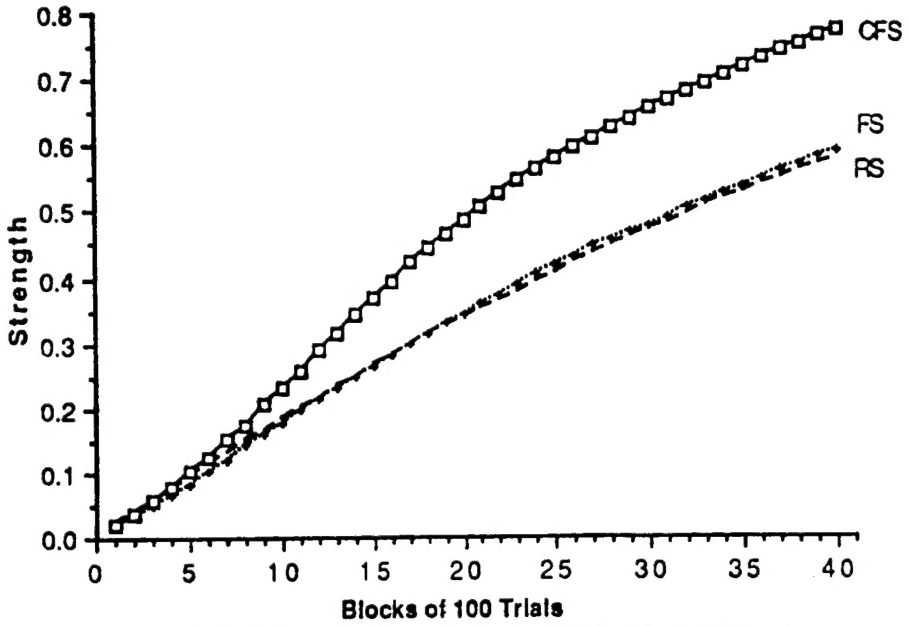


Figure 4: Learning for Examples with Multiple Clusters

Discussion

Attention can serve as an important learning bias for learning by observation. The type of attention mechanism used, however, should be sensitive to context if the training examples are complex, i.e., have multiple clusters. Contextually Focused Sampling (CFS) was proposed to be a better match to human attentional processes than earlier models that were not sensitive to context (Billman & Heit, 1988). CFS is also an important candidate for an attentional bias in machine learning systems.

These computational experiments suggest several interesting predictions about human learning. First, if humans use either FS or CFS, then they must show faster learning than random sampling (RS) would permit. Second, if humans use CFS and not FS, they should demonstrate faster learning of multiple clusters than either FS or RS would permit. Finally, CFS would predict that humans would show different probabilities of sampling particular features depending upon what they had already sampled.

For machine learning, the results imply that CFS can be used to make induction more feasible. Additionally, the results of Experiment III suggest how CFS might be used to divide a set of training examples into appropriate categories of interpredictive features. Because CFS is capable of finding and learning about multiple subclusters of interrelated features, it can provide a method for constructing a hierarchy of probabilistic concepts. The use of focused attention to easily isolate these concepts may result in an algorithm that is more efficient and more powerful than current concept learning methods (Fisher, 1987). Such concept acquisition would also allow a CFS system to learn higher order conditional probabilities to improve its inference capabilities (Chalnick & Billman, 1988; Davis, 1985).

References

Anderson, J. R. (1988). The place of cognitive architectures in a rational analysis. In *Proceedings of Tenth Annual Conference of the Cognitive Science Society.*

Billman, D. & Heit, E. (in press). Observational learning from internal feedback: a simulation of an adaptive learning method. *Cognitive Science.*

Billman, D., Heit, E., & Dorfman, J. (1987). Facilitation from clustered features: using correlations in observational learning. In *Proceedings of Ninth Annual Conference of the Cognitive Science Society.*

Chalnick, A. & Billman, D. (1988). Unsupervised learning of correlational structure. In *Proceedings of Tenth Annual Conference of the Cognitive Science Society.*

Davis, B. R. (1985). An associative hierarchical self-organising system. *IEEE Transactions on Systems, Man, and Cybernetics, 15,* 570–579.

Fisher, D. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning, 2,* 139–172.

Loftus, G. R. & Mackworth, N. H. (1978). Cognitive determinants of fixation location during picture viewing. *Journal of Experimental Psychology: Human Performance and Perception, 4,* 565–572.

Martin, J. D. (1988). Cora: a best match memory for case storage and retrieval. In *Proceedings of AAAI Case-Based Reasoning Workshop.*

Medin, D. L. & Schaffer, M. M. (1978). A context theory of classification learning. *Psychological Review, 85,* 207–238.

Rendell, L., Seshu, R., & Tcheng, D. (1987). More robust concept learning using dynamically-variable bias. In *Proceedings of the Fourth International Workshop on Machine Learning.*

Rescorla, R. A. (1972). Informational variables in pavlovian conditioning. In G. H. Bower (Ed.), *The Psychology of Learning and Motivation.* New York: Academic Press.

Rosch, E. H. (1978). Principles of categorisation. In E. H. Rosch & B. B. Lloyd (Eds.), *Cognition and Categorization.* Hillsdale, NJ: Erlbaum.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propogation. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel Distributed Processing, Vol 1.* Cambridge, Massachusetts: MIT Press.

Schlimmer, J. C. (1987). Incremental adjustment of representations for learning. In *Proceedings of the Fourth International Workshop on Machine Learning.*

Trabasso, T. & Bower, G. H. (1968). *Attention in Learning.* New York: Wiley.

Zeaman, D. & House, B. J. (1963). The role of attention in retardate discrimination learning. In N. R. Ellis (Ed.), *Handbook of Mental Deficiency.* New York: McGraw Hill.